Proceedings of the 2018 IEEE Conference on Computational Intelligence and Games (CIG'18)

Cameron Browne Proceedings Chair
Mark H. M. Winands General Chair
Jialin Liu Program Chair
Mike Preuss Program Chair

Department of Data Science & Knowledge Engineering
Maastricht University
Maastricht, The Netherlands

14–17 August, 2018

Copyright

2018 IEEE Conference on Computational Intelligence and Games (CIG'18).

IEEE Catalog Number: CFP18CIG-ART
ISBN: 978-1-5386-4359-4
ISSN: 2325-4289

Additional copies of this publication are available from:

Curran Associates, Inc.,
57 Morehouse Lane,
Red Hook, NY 12571, USA
+1 845 758 0400
+1 845 758 2633 (FAX)
Email: curran@proceedings.com

Preface

These Proceedings contain the papers presented at the 2018 IEEE Conference on Computational Intelligence and Games (CIG'18) held in Maastricht, the Netherlands, over 14–17 August 2018.

The annual Computational Intelligence and Games (CIG) conference series brings together leading researchers and practitioners from academia and industry, to discuss recent advances in the field and explore future research directions, and is one of the premier international conferences in this exciting and expanding field. Earlier CIG conferences took place in Colchester (UK), Reno (USA), Honolulu (Hawaii), Perth (Australia), Milan (Italy), Copenhagen (Denmark), Seoul (South Korea), Granada (Spain), Niagara Falls (Canada), Dortmund (Germany), Tainan (Taiwan), Santorini (Greece) and New York (USA).

For this 14^{th} edition of the conference, we received a total of 110 papers from 36 countries. All papers were peer reviewed by at least three domain experts, and 51/110 were accepted for oral presentation (acceptance rate $46.\dot{3}\dot{6}\%$). A further 18 papers were accepted as posters. The program includes five vision papers, seven competition papers and one demo paper.

These proceedings would not have been possible without the help of many individuals. In particular, we would like to thank the authors and the Programme Committee members for their help. Finally, the editors recognise the generous sponsors for this event: MaTHiSiS, NWO, SWOL and the city of Maastricht.

1 July 2018
Maastricht,
The Netherlands

Cameron Browne
Mark Winands
Jialin Liu
Mike Preuss

Organizing Committee

General Chair
Mark H. M. Winands (Maastricht University)

Program Chairs
Yngvi Björnsson (Reykjavík University)
Jialin Liu (Southern University of Science and Technology)
Mike Preuss (ERCIS, WWU Münster)

Proceedings Chair
Cameron Browne (Maastricht University)

Tutorial and Keynote Chair
Stylianos (Stelios) Asteriadis (Maastricht University)

Special Session Chair
Santiago Ontañón (Drexel University)

Competition Chair
Diego Perez-Liebana (Queen Mary University of London)

Finance Chair
Gerasimos Spanakis (Maastricht University)
Kostas Karpouzis (National Technical University of Athens)

Publicity Chair
Mirela Popa (Maastricht University)

Local Arrangement Chairs
Chiara Sironi (Maastricht University)
Enrique Hortal (Maastricht University)

Webmaster
Christos Athanasiadis (Maastricht University)

Table of Contents

Programme Committee

Samad Ahmadi	De Montfort University
Vincent Aleven	Carnegie Mellon University
Daniel Ashlock	University of Guelph
Hamna Aslam	Innopolis University
Stylianos Asteriadis	Maastricht University
Davide Aversa	Sapienza University of Rome
Byung-Chull Bae	Hongik University
Hendrik Baier	Digital Creativity Labs, University of York
Ryan Baker	University of Pennsylvania
Sander Bakkes	Tilburg University
Christian Bauckhage	Fraunhofer
Amit Benbassat	Ben-Gurion University of the Negev
Rafael Bidarra	Delft University of Technology
Yngvi Björnsson	Reykjavik University
Alan Blair	The University of New South Wales
Philip Bontrager	New York University
Bruno Bouzy	Paris Descartes University
Kristy Boyer	University of Florida
Ivan Bravi	Queen Mary University of London
Keith Brawner	United States Army Research Laboratory
Joost Broekens	Delft University of Technology
Joseph Alexander Brown	Innopolis University
Cameron Browne	Maastricht University
Phillip Buffum	North Carolina State University
Paolo Burelli	Aalborg University
Alessandro Canossa	Northeastern University
Marc Cavazza	University of Greenwich
Tristan Cazenave	LAMSADE Université Paris Dauphine PSL CNRS
Massimiliano Cazzaniga	imaginary
Fred Charles	Bournemouth University, Faculty of Science and Technology
Ioannis Chatzigiannakis	Sapienza University of Rome
Ran Cheng	University of Surrey
Yun-Gyung Cheong	SKKU
Sung-Bae Cho	Yonsei University
David Churchill	University of Alberta
Simon Colton	Goldsmiths College, University of London
Seth Cooper	Northeastern University
Peter Cowling	University of York
Maria Cutumisu	University of Alberta
Tasos Dagiuklas	London South Bank University
Vanessa De Luca	Supsi - LCV - Interaction Design Lab
Joerg Denzinger	University of Calgary
Sam Devlin	Microsoft
Alexander Dockhorn	OvGU Magdeburg
Michael Eagle	Carnegie Mellon University
Antonio Fernández Ares	University of Granada

Antonio J. Fernández Leiva	Universidad de Málaga
Vilma Ferrari	Institute of Mobile Technologies for Education and Culture
J. T. Folsom-Kovarik	Soar Technology, Inc.
Allan Fowler	Kennesaw State University
Raluca Gaina	Queen Mary University of London
Marcus Gallagher	The University of Queensland
Jeremy Gow	Goldsmiths, University of London
Kazjon Grace	The University of Sydney
Daniele Gravina	University of Malta
Garry Greenwood	Portland State University
Christian Guckelsberger	Goldsmiths College, University of London
Mathew Guzdial	Georgia Institute of Technology
Hisashi Handa	Kindai University
Casper Harteveld	Northeastern University
Tomonori Hashiyama	The University of Electro-Communications
Erin Hastings	University of Central Florida
Ami Hauptman	Sapir College
Rania Hodhod	Columbus State University
Christoffer Holmgård	Northeastern University
Amy K. Hoover	New Jersey Institute of Technology
Ian Horswill	Northwestern University
Hiroyuki Iida	JAIST
Aaron Isaksen	New York University
Hisao Ishibuchi	Osaka Prefecture University
Mikhail Jacob	Georgia Institute of Technology
Anna Jordanous	University of Kent
Niels Justesen	IT University of Copenhagen
Sotiris Karachontzitis	University of Patras
Daniel Karavolos	University of Malta
Konstantinos Karpouzis	National Technical University of Athens
Ahmed Khalifa	New York University
Kyung-Joong Kim	Sejong University
Sven Koenig	University of Southern California
Stefanos Kollias	Independent
Aris Lalos	Electrical & Computer Engineering Dept.
Marc Lanctot	Google
H. Chad Lane	University of Illinois at Urbana-Champaign
Pierluca Lanzi	Politecnico di Milano
Joel Lehman	University of Central Florida
Helen Leligou	Piraeus University of Applied Sciences
James Lester	North Carolina State University
John Levine	University of Strathclyde
Antonios Liapis	Institute of Digital Games, University of Malta
Jialin Liu	Southern University of Science and Technology
Siming Liu	University of Nevada, Reno
Daniele Loiacono	Politecnico di Milano
Phil Lopes	Institute of Digital Games, University of Malta
Sushil Louis	University of Nevada, Reno

Simon Lucas	Queen Mary University of London
Ioanna Lykourentzou	Utrecht University
Collin Lynch	North Carolina State University
Dario Maggiorini	University of Milan
Ilias Maglogiannis	University of Piraeus
Tobias Mahlmann	Moon Mill Studios
Akis Makris	Institute of Communication and Computer Systems
Manolis Maragoudakis	University of the Aegean
Carlos Martinho	Universidade de Lisboa
Helmut Mayer	University of Salzburg
Jacek Mańdziuk	Warsaw University of Technology
Wookhee Min	North Carolina State University
Antonio Mora	University of Granada
Bradford Mott	North Carolina State University
Hector Munoz-Avila	Lehigh University
Phivos Mylonas	National Technical University of Athens
Andy Nealen	New York University
Mark Nelson	Falmouth University
James Niehaus	Charles River Analytics
Benjamin Nye	University of Southern California
Santiago Ontanon	Drexel University
Diego Perez Liebana	Queen Mary University of London
Ana Maria Pinuela Marcos	Independent
Andrew Pomazanskyi	Nurogames GmbH
Mike Preuss	WWU Muenster
William Raffe	University of Technology, Sydney
Robert Reynolds	Wayne State University
Sebastian Risi	IT University of Copenhagen
Ma Mercedes Rodrigo	Ateneo de Manila University
Elizabeth Rowe	TERC
Jonathan Rowe	North Carolina State University
Guenter Rudolph	TU Dortmund University
Thomas Runarsson	University of Iceland
Owen Sacco	Institute of Digital Games, University of Malta
Yago Saez	University Carlos III of Madrid
Christoph Salge	University of Hertfordshire
Spyridon Samothrakis	University of Essex
Björn Schuller	University of Augsburg / Imperial College London
Moshe Sipper	Ben-Gurion University of the Negev
Chiara F. Sironi	Maastricht University
Adam M. Smith	University of California Santa Cruz
Sam Snodgrass	Northeastern University
Gerasimos Spanakis	Maastricht University
Pieter Spronck	Tilburg University
Anastasios Tefas	Aristotle University of Thessaloniki
Fabien Teytaud	Laboratoire LISIC, Université du Littoral Cote d'Opale
Ruck Thawonmas	Ritsumeikan University
Thomas Thompson	Table Flip Games Ltd.

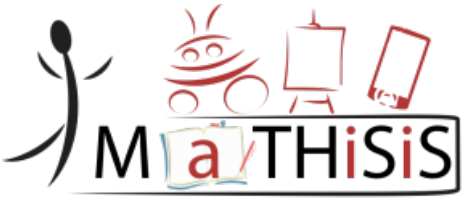
Frank Thuijsman	Maastricht University
Julian Togelius	New York University
Laurissa Tokarchuk	Queen Mary University of London
Ruben Torrado	New York University
José Valente de Oliveira	Universidade do Algarve
Vanessa Volz	TU Dortmund University
Ning Wang	University of Southern California
Mark Winands	Maastricht University
Kevin Wong	Murdoch University
John Woodward	Queen Mary University of London
I-Chen Wu	National Chiao Tung University
Georgios N. Yannakakis	Institute of Digital Games, University of Malta
Fabio Zambetta	RMIT University
Diego Zapata-Rivera	Educational Testing Service
Alexander Zook	Georgia Institute of Technology

Additional Reviewers

Bravi, Ivan
Gaber, Tarek
Gravina, Daniele
Green, Michael C
Karavolos, Daniel
Li, Jiaoyang
Mavromoustakos-Blom, Paris
Melhart, David
Moerland, Thomas
Nygren, Nick
Pezzullo, Lydia
Wilkinson, Jamieka

Sponsors

We would like to thank the following sponsors for their generous contributions to CIG'18.



Managing Affective-learning THrough Intelligent atoms and Smart InteractionS

University Fund Limburg (SWOL)

City of Maastricht

Shallow Decision-Making Analysis in General Video Game Playing

Ivan Bravi, Diego Perez-Liebana and Simon M. Lucas
School of Electronic Engineering and Computer Science
Queen Mary University of London
London, United Kingdom
{i.bravi, diego.perez, simon.lucas}@qmul.ac.uk

Jialin Liu
Southern University of Science and Technology
Shenzhen, China
liujl@sustc.edu.cn

***Abstract*—The General Video Game AI competitions have been the testing ground for several techniques for game-playing, such as evolutionary computation techniques, tree search algorithms, hyper-heuristic-based or knowledge-based algorithms. So far the metrics used to evaluate the performance of agents have been win ratio, game score and length of games. In this paper we provide a wider set of metrics and a comparison method for evaluating and comparing agents. The metrics and the comparison method give shallow introspection into the agent's decision-making process and they can be applied to any agent regardless of its algorithmic nature. In this work, the metrics and the comparison method are used to measure the impact of the terms that compose a tree policy of an MCTS-based agent, comparing with several baseline agents. The results clearly show how promising such general approach is and how it can be useful to understand the behaviour of an AI agent, in particular, how the comparison with baseline agents can help understanding the shape of the agent decision landscape. The presented metrics and comparison method represent a step toward to more descriptive ways of logging and analysing agent's behaviours.**

***Index Terms*—artificial general intelligence, general video game play, game-playing agent analysis, game metrics**

I. Introduction

General video game playing (GVGP) and General game playing (GGP) aim at designing AI agents that are able to play more than one (video) game successfully alone without human intervention. One of the early stage challenges is to define a common framework that allows the implementation and testing of such agents on multiples games. For this purpose, the General Video Game AI (GVGAI) framework [1] and General Game Playing framework [2], [3] have been developed. Competitions using the GVGAI and GGP frameworks have significantly promoted the development of a variety of AI methods for game-playing. Examples include tree search algorithms, evolutionary computation, hyper-heuristic, hybrid algorithms, and combinations of them. GVGP is more challenging due to the possibly stochastic nature of the games to be played and the short decision time. Five competition tracks have been designed based on the GVGAI framework for specific research purposes. The planning and learning tracks focus on designing an agent that is capable of playing several unknown games respectively with or without the forward model to simulate future game states. The level and rule generation tracks have the objective of designing AI programs that are capable of creating levels or rules based on a game specification. Despite the fact that the initial purpose of developing GVGAI framework was to facilitate the research on GVGP, GVGAI and its game-playing agents have also been used in other application rather than just competitive GGP. For instance, the GVGAI level generation track has used the GVGAI game playing agents to evaluated the automatically generated game levels. Relative algorithm performance [4] has been used to understand how several agents perform in the same level. Although, no introspection into the agent behaviour or decision-making process was used so far.

The main purpose of this paper is to give a general set of metrics that can be gathered and logged during the agent's decision-making process to understand its in-game behaviour. These are meant to be generic, shallow and flexible enough to be applied to any kind of agent regardless of its algorithmic nature. Moreover we are also providing a generic methodology to analyse and compare game-playing agents in order to get an insight on how the decision-making process is carried out. This method will be later addressed as *comparison method*.

Both the metrics and the comparison method will be useful in several applications. It can be used for level generation: knowing the behaviour of an agent and what attracts it in the game-states space means that it can be used to measure how a specific level design suits a certain play-style therefore pushing the design to suit the agent in a recommender system fashion [5]. From a long term perspective, this can be helpful to understand a human player's behaviour and then personalise a level or a game to meet this player's taste or playing style. Solving the dual problem is useful as well, in the process of looking for an agent that can play well a certain level design, disposing of reliable metrics to analyse the agent behaviour could significantly speed up the search. Additionally, by analysing the collected metrics, it's possible to find out if a rule or an area of the game world is obsolete. This can be also applied generally to the purpose of understanding game-playing algorithms, it's well known that there are black-box machine learning techniques that offer no introspection in their reasoning process, thus being able of comparing in a shallow manner, the decision-making process of different agents can help shed some light into their nature. A typical example is a neural network that given some input features outputs the action probability vector. With the proposed metrics and methodology it would be possible to

978-1-5386-4359-4/18/$31.00 ©2018 IEEE

make estimate its behaviour without actually looking at the agent playing the game and extracting behavioural information by hand.

The rest of this paper is structured as follows. In Section II, we provide a background on the GVGAI framework focusing in particular on the game-playing agents, three examples of how agent performance metrics have been used so far in scenarios other than pure game-play and an overview of MCTS-based agents. Then, we propose a comparison method, a set of metrics and an analysis procedure in Section III. Experiments using these metrics are described in Section IV and the results are discussed in Section V to demonstrate how they provide a deeper understanding on the agent's behaviour and decision-making. Last, we draw final considerations and list possible future work in Section VI.

II. Background

A. General Video Game AI framework

The General Video Game AI (GVGAI) framework [1] has been used for organising GVGP competitions at several international conferences on games or evolutionary computation, for research and education in worldwide institutions. The main GVGAI framework is implemented using *Java* and *Python*. A Python-style Video Game Description Language (VGDL) [6], [7] is developed to make it possible to create and add new games to the framework easily. The framework enables several tracks with different research purposes. The objective of the single-player [8] and two-player planning [9] tracks is to design an AI agent that is able to play several different video games respectively alone or with another agent. With access to the current game state and the forward model of the game, a planning agent is required to return a legal action in a limited time. Thus, it can simulate games to evaluate an action or a sequence of actions and get the possible future game state(s). However, in the learning track, no forward model is given, a learning agent needs to learn in an trial-and-error way. There are two other tracks based on the GVGAI framework which focus more on game design: the rule generation [10] and the level generation [11]. In the rule generation track, a competition entry (generator) is required to generate game rules (interactions and game termination conditions) given a game level as input, while in the level generation track, an entry is asked to generate a level for a certain game. The rule generator or level generator should be able to generate rules or levels for any game given a specified search space.

B. Monte Carlo Tree Search-based agents

Monte-Carlo Tree Search (MCTS) has been the state-of-the-algorithm in game playing [12]. The goal of MCTS is to approximate the value of the actions/moves that may be taken from the current game state. MCTS builds iteratively a search tree using Monte Carlo sampling in the decision space and the selection of the node (action) to expand is based on the outcome of previous samplings and on a Tree Policy. A classic Tree Policy is the Upper Confidence Bound (UCB) [13]. The UCB is one of the classic multi-armed bandit algorithms which aims at balancing between exploiting the best-so-far arm and exploring more the least pulled arms. Each arm has an unknown reward distribution. In the game-playing case, each arm models a legal action from the game state (thus a node in the tree), a reward can be the game score, a win or lose of a game, or a designed heuristic. The UCB Tree Policy selects to play the action (node) a^* such that $a^* = \arg\max_{a \in A} \ \bar{x}_a + \sqrt{\frac{\alpha \ln n}{n_a}}$, where A denotes the set of legal actions at the game state, n and n_a refers to the total number of plays and the number of times that the action a has been played (visited), α is called exploration factor.

The GVGAI framework provides several sample controllers for each of the tracks. For instance, the *sampleMCTS* is a vanilla implementation of MCTS for single-player games, but performs finely on most of the games. M. Nelson [14] tests the *sampleMCTS* on more than sixty GVGAI games, using different amounts of time budget for planning at every game tick, and observes that this implementation of MCTS is able to reduce the loss rate given longer planning time. More advanced variants of MCTS have been designed for playing a particular game (e.g., the game of Go [15], [16]), for general video game playing (e.g., [8], [17]) or general game playing (e.g., [18]). Recently, Bravi *et al.* [19] custom various heuristics particularly for some GVGAI games, and Sironi *et al.* [20] design several Self-Adaptive MCTS variants which use hyper-parameter optimisation methods to tune on-line the exploration factor and maximal roll-out depth during the game playing.

C. Agent performance evaluation

Evaluating the performance of an agent is sometimes a very complex task depending on how the concept of performance is defined. In the GVGAI planning and learning competitions, an agent is evaluated based on the the amount of games it wins over a fixed number of trials, the average score that it gets and the average duration of the games. Sironi *et al.* [20] evaluate the quality of their designed agents using a heuristic which combines the score obtained eventually giving an extra bonus or penalty depending on whether the agent could reach a winning state or a losing state, respectively. The GVGAI framework has also been used for purposes other than the ones laid out by the competition tracks. Bontrager *et al.* [21] cluster some GVGAI single-player and two-player games using game features and agent performance extracted using the playing data by the single-player and two-player planning competition entries, respectively. In particular, the performance of an agent, represented by win ratio in [21], is used to cluster the games in four groups: games easy to win, hard games, games that MCTS agent can play well and games that can be won by a specific set of agents. The idea behind that work is interesting although the clustering results in three small sized groups and a very large one. This suggests that using more introspective metrics could help clustering the games more finely.

GVGAI has also been used as test bed for evolving MCTS tree policies (in the form of a mathematical formula for decision making) for specific games [19]. [19] consists in evolving Tree Policies (formulae) using Genetic Programming,

the fitness evaluation is based on the performance of an MCTS agent which uses the specific tree policy. Once again, the informations logged and used from the playthrough by the fitness function were a combination of win ratio, average score and average game-play time, in terms of the number of game ticks. Unfortunately no measurement was made about the robustness of the agent's decision-making process of which could have been embedded in the fitness function to possibly enhance the evolutionary process. In the recent Dagstuhl seminar on AI-Driven Game Design, game researchers have envisioned a set of features to be logged during game-play, divided into four main groups: direct logging features, general indirect features, agent-based features and interpreted features [22]. A preliminary example of how such features can be extracted and logged in the GVGAI framework has also been provided [22]. Among the direct logging features, we can find some kind of game information that don't need any sort of interpretation, few examples are: game duration, actions log, game outcome and score. Instead, these features are listed in the general indirect features which require some degree of interpretation or analysis of the game state such as the entropy of the actions, the game world and the game state space. The agent-based features gather information about the agent(s) taking part to the game, for example about the agent surroundings, the exploration of the game-state space or the convention between different agents. Finally, the interpreted features are based on metrics already defined in previous works such as drama and outcome uncertainty [23] or skill depth [24].

III. Methods

This section first introduces a set of metrics that can potentially be extracted from any kind of agent regardless of its algorithmic nature, aiming at giving an introspection of the decision-making process of a game-playing agent in a shallow and general manner (Section III-A). Then we present a method to compare the decisions of two distinct game-playing agents under identical conditions using the metrics introduced previously. As described in [25] the decision-making comparison can be done at growing levels of abstraction: action, tactic or strategic level. Our proposed method compares the decision-making at the action level. Later, we design a scenario in which the metrics and the comparison method are used to analyse the behaviour of instances of an MCTS-agent using different tree policies comparing them to agents with other algorithmic natures. Finally we describe the agents used in the experiments.

In this paper, the following notations are used. A *playthrough* refers to a complete play of a game from beginning to end. The set of available actions is denoted as $\mathcal{A}$ being $N = |\mathcal{A}|$, a_i refers to the i^{th} action in $\mathcal{A}$. A *budget* or *simulation budget* is either the amount of forward-model calls the agent can make at every game tick to decide the next action to play or the CPU-time that the agent can take. The fixed budget is later addressed as $\mathcal{B}$.

A. Metrics

The metrics presented in this paper are based on two simple and fairly generic assumptions: (1) for each game tick the agent considers each available action a_i for n_i times; (2) for each game tick the agent assigns a value $v(a_i)$ to each available action. In this scenario the agents are designed to operate on a fixed budget $\mathcal{B}$ in terms of real time or number of forward model calls, which allows for a fair comparison making the measurements comparable between each other.

Due to the stochastic nature of an agent or a game, it is sometimes necessary to make multiple playthroughs for evaluation. The game id, level id, outcome (specifically, win/loss, score, total game ticks) and available actions at every game tick are logged for each playthrough. Additionally, for each game tick in the playthrough, the agent is going to provide the following set of metrics:

- a^*: the recommended action to be played next;
- $\overline{p}$: probability vector where p_i represents the probability of considering a_i during the decision-making process;
- $\overline{v}$: vector of values $v_i \in \mathcal{R}$ where v_i is the value of playing a_i from the current game state, v^* is the highest value which implies it being associated with a^*. Whenever the agent doesn't actually have such information about the quality of a_i then v_i should be NaN;
- b: represents the ratio of the budget consumed over the fixed available budget $\mathcal{B}$, $b \in [0, 1]$ where 0 and 1 respectively mean that either no budget or the whole $\mathcal{B}$ was used by the agent;
- $conv$: convergence, as the budget is being used is likely for the current a^* to fluctuate, $conv$ is the ratio of budget used over $\mathcal{B}$ when a^* is stable. It means that any budget used after $conv$ hasn't changed the recommended action. $conv \in [0, b]$.

It is notable that most of the agents developed for the GVGAI try to consume as much budget as possible, however this is not necessarily a good trait of the agent, being able to log the amount of budget used and distinguish between a budget-saver and a budget-waster can give an interesting insight on the decision-making process especially on the confidence of the agent. Since this set of metrics tries to be as generic as possible, we shouldn't limit the metrics because of the current agent implementations. The vectors $\overline{p}$ and $\overline{v}$ can be inspected to portray the agent preference over $\mathcal{A}$. The vector $\overline{p}$ can also be used during the debug phase of designing an agent to see whether it actually ever considers all the available action.

Generally different agents reward actions differently, therefore it is not possible to make a priori assumptions on the range or the distribution over values. Although the values in $\overline{v}$ allow at the very least to rank the actions and moreover to get informations about their boundaries and distributions (guaranteed a reasonable amount of data) a posteriori. Furthermore, it is possible to follow the oscillation of such values through the game-play highlighting critical portions of it. For example, when the v_i are similar (not very far apart from each other

considering the value bounds logged) and generally high then we can argue that the agent evaluates all actions as good ones. On the contrary if the values are generally low, the agent is probably struggling in a bad game scenario.

B. Comparison method

Comparing the decisions made by different agents is not a trivial matter especially when their algorithmic nature can be very different. The optimal set-up under which we can compare their behaviour is when they are provided the same problem or scenario under exactly same conditions. This is sometimes called *pairing*. We propose the following experimental set-up: a meta-agent, called *Shadowing Agent*, instantiates two agents: the *main* agent, and the *shadow* agent. For each game tick the *Shadowing Agent* behaves as a proxy and feeds the current game state to each of the agents which will provide the next action to perform as if it was a normal GVGAI game-play execution. Both these agents have a limited budget. Once both main and shadow agent behaviours are simulated, the *Shadowing Agent* takes care of logging the metrics described previously from both agents and then returns to the framework the action chosen by the main agent. In this way the actual avatar behaviour in the game simulated is consistent with the main agent and the final outcome represents its performance. In the next sections we are going to use the superscripts ^{m} and ^{s} for a metric respectively relative to the main agent or the shadow agent. A typical scenario would be comparing how very radically different agents such as: a Random agent, a Monte-Carlo Search agent, a One-Step Look Ahead agent and an MCTS-based agent. Under this scenario, comparing each single coupling of agents will result in producing a matrix of comparisons. All the informations on how the agents extract the metrics described previously are detailed in Section IV-B.

C. Analysis Method

We are going to analyse these agents' behaviours in few games, for each game we are going to run all the possible couplings of main agent and shadow agent, for each couple we are going to run N_p playthroughs and, finally, for each playthrough we are going to save the current metrics for both main and shadow agents. It's worth remembering that each playthrough has its own length, thus playthrough i will have length l_i. This means that in order to analyse and compare behaviours we need a well structured methodology to slice data appropriately. Our proposed method is represented in Figure 1. The first level of comparison is done at the action level, we can measure two things: *Agreement Percentage* $\mathcal{AP}$, percentage of times the agents agreed on the best action averaged across the several playthroughs; and *Decision Similarity* $\mathcal{DS}$, the average symmetric Kullback-Leibler divergence of the two probability vectors $\overline{p}^m$ and $\overline{p}^s$. When $\mathcal{AP}$ is close to 100% or $\mathcal{DS} \sim 0$ we have two agents with similar behaviours, at this point we can step to the next level of comparison: *Convergence*, we compare $conv^m$ and $conv^s$ to see if there is a faster converging agent; and *Value Estimation*, this level of comparison is thorny, in fact each agent has its own function for evaluating a possible action, for this step we recommend using these values to rank the actions using them as preference evaluation. *Convergence* can highlight both the ambiguity of the surrounding game states or the inability of the agent to recognise important features. If the agents have a similar $conv$ values we can then take a look at the *Efficiency*. This value represents the average amount of budget used by the agent.

To summarise, once two agents with similar $\mathcal{AP}$ or $\mathcal{DS}$ are found, the next comparison levels highlight the potential preference toward the fastest converging and most budget-saver one.

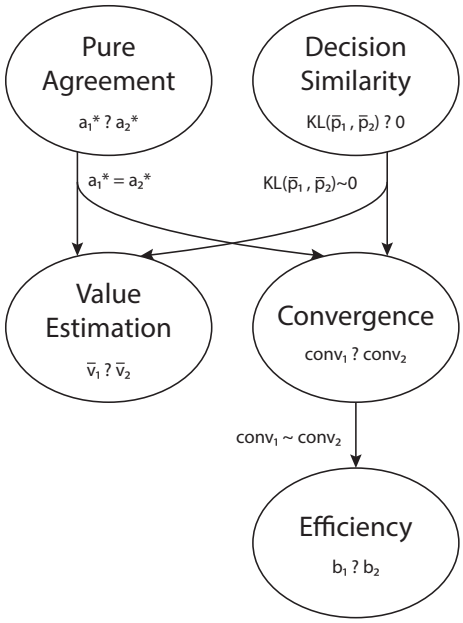


Fig. 1: The decision graph to compare agents' behaviours.

IV. Experimental set-up

In this section, we show how a typical experiment could be run using the metrics and methods introduced previously. Each experiment is run over the following games in order to have diverse scenarios that can highlight different behaviours:

- Aliens: a game loosely modelled on the Atari 2600's Space Invaders, the agent on the bottom of the screen has to shoot the incoming alien spaceships from above avoiding their blasts;
- Brainman: the objective of the game is for the player to reach the exit, the player can collect diamonds to get points and push keys into doors to open them;
- Camel Race: the player, controlling a camel, has to reach the finish line before the other camels whose behaviour is part of the design of the game;
- Racebet: in the game there are few camels racing toward the finish line, each has a unique colour, in order to win the game the agent has to position the avatar on the camel with a specific colour;

- Zenpuzzle: the level has two different types of floor tiles, one that can be always stepped on and a special type that can be stepped on no more than once. The agent has to step on all the special tiles in order to win the game.

Further details on the games and the framework can be found at www.gvgai.net . The budget given to the agents is a certain number of forward-model calls which is different than the real time constraints used in the GVGAI competitions. We made this decision in order to get more robust data across different games, in fact the number of forward model calls that can be executed in the 40 ms can drastically vary changing the game, sometimes from hundreds to thousands.

This experiment consists in running the comparisons between the MCTS-based agents that use all possible prunings $h' \in \mathcal{H}$ as tree policy generated from h (cf. (1), variables summarised in Table I), and the following agents: Random, One-Step Look Ahead, and Monte-Carlo Search.

$$h = min(D_{MOV}) \cdot min(D_{NPC}) + \frac{|max(R)|}{\sum D_{NPC}} \quad (1)$$

In this work, each pair of agents is tested over 20 playthroughs

TABLE I: Variables used in the heuristic (cf. (1)).

Notation	Description
$max(R)$	Highest reward among the simulations that visit current node
$min(D_{MOV})$	Minimum distance from a movable sprite
$min(D_{NPC})$	Minimum distance from an NPC
$sum(D_{NPC})$	Sum of all the distances from NPCs

of the first level of each game, all the agents were given a budget of 700 forward-model calls. The budget was decided looking at the average number of forward-model calls done in all the GVGAI games by the Genetic Programming MCTS (GPMCTS) agent with a time budget of 40 ms, same as in the competitions. The GPMCTS agent is an MCTS agent with customisable Tree Policy as described in [19].

A. Comparison method for MCTS-based agents

MCTS-based agents can be tuned and enhanced in many different ways, a wide set of hyper-parameters can be configured differently, one of the most crucial components is the tree policy. The method we propose gradually prunes the tree policy heuristic in order to isolate bits of (1). Evaluating the similarity of two tree policies is a rather complex task, it can be roughly done by analysing the difference between their values given a point in their search domain. This approach is not optimal, supposing we want to analyse two functions f and g where $g = f + 10$, their values will never be the same but when applied to the MCTS scenario they would perform exactly the same. Actually, what matters is not the exact value of the function but the way that two points in the domain are ordered according to their evaluations. In short, being $\mathcal{D}$ the domain of the functions f and g and $p_1, p_2 \in \mathcal{D}$ what matters is that both the following conditions $f(p_1) \geq f(p_2)$ and $g(p_1) \geq g(p_2)$ hold true. The objective is to understand how each term in (1) used in the tree policy of an MCTS agent impacts the behaviour of the whole agent. Given h, thus (1) used as tree policy, let $\mathcal{H}$ be the set of all possible prunings (therefore functions) of the expression tree associated to h. This method applies the metrics and the comparison method introduced previously and it consists in running all possible couples $(A_m, A_s) \in \mathcal{AG} \times \mathcal{AG}$ where the agent A_m is the main agent and A_s is the shadow agent, the set $\mathcal{AG}$ contains one instance of MCTS-based agent for each tree policy in $\mathcal{H}$ and the following agents: Random, One-Step Look Ahead, Monte-Carlo Search. In this way it is possible to get a meaningful evaluation of how different equations might result in suggesting the same action, or not, for all the possible comparisons of the equations in $\mathcal{H}$ but also how they compare to the other reference agents.

B. Agents

In this section, we give the specifications of the agents used and the way they link each metric to their algorithmic implementation. These agents are going to be used in the experiments and they can be used as examples of how algorithmic informations can be interpreted and manipulated to get the metrics described previously. Most agents use *SimpleStateHeuristic* which evaluates a game state according to the win/lose state, the distance from portals and the number of NPCs. It rewards best winning states with no NPCs and where the position of the player is closest to a portal. None of the agents was chosen for its performance, the point of using these agents is that theoretically they can represent very different play styles: completely stochastic, very short-sighted, randomly long-sighted, generally short-sighted.

1) Random: The random agent has a very straightforward implementation: given the set of available actions, it picks an action uniformly at random.

- $\overline{p}$: since the action is picked uniformly $p_i = 1/|\mathcal{A}|$;
- $\overline{v}$: each v_i is set to NaN;
- $b = 0$, since no budget is consumed to return a random action;
- $conv$ is always 0 for the same reason of b.

2) One-Step Look Ahead: The agent makes a simulation for each of the possible actions, and evaluates the resulted game state using the *SimpleStateHeuristic* defined by the GVGAI framework. The action with the highest values is going to be picked as a^*.

- $\overline{p}$: $p_i = 1/|\mathcal{A}|$ since each action is picked once;
- $\overline{v}$: each v_i corresponds to the evaluation given by the *SimpleStateHeuristic* initialized with current game state and compared to the game state reached via action a_i;
- b is always $\frac{|\mathcal{A}|}{sb}$;
- $conv$ varies and corresponds to the budget ratio when the best action is simulated.

3) Monte-Carlo Search: The Monte-Carlo Search agent performs a Monte-Carlo sampling of the action-sequence space following 2 constraints: the sequence is not longer than 10 and only the last action can bring to a termination state.

- $\overline{p}$: considering n_i as the number of times action a_i was picked as first action and $N = \sum_{i=0}^{|\mathcal{A}|} n_i$ then $p_i = \frac{n_i}{N}$;
- $\overline{v}$: each v_i is the average evaluation by the *SimpleStateHeuristic* initialized with the current game state compared to each last game state reached by every action sequence started from a_i;
- b is always 1, since the agent keeps simulating until the end of the budget;
- $conv$ corresponds to the ratio of budget used at the moment the action with the highest v_i last changed.

4) MCTS-based: The MCTS-based is an implementation of MCTS with uniformly random roll-outs to a maximum depth of 10. The tree policy used can be specified when the agent is initialised, therefore the reader should not suppose UCB1 as the tree policy, whereas the heuristic used to evaluate game states is a combination of the score plus an eventual bonus/penalty for a win/lose state.

- $\overline{p}$: considering n_i as the number of visits for a_i at the root node of the search tree and N as the number of visits at the root node then $p_i = \frac{n_i}{N}$;
- $\overline{v}$: each v_i is the heuristic value associated to a_i at the root node;
- $b = 1$, since the agent keeps simulating until the budget is used up;
- $conv$ corresponds to the ratio of budget used when the action with the highest v_i last changed in the root node.

V. Experiments

TABLE II: Agents used in experiments and their ids.

Id	Agent
0	MCTS + $\frac{1}{\sum D_{NPC}}$
1	MCTS + $\|max(R)\|$
2	MCTS + $\frac{\|max(R)\|}{\sum D_{NPC}}$
3	MCTS + $min(D_{NPC})$
4	MCTS + $min(D_{NPC}) + \frac{1}{\sum D_{NPC}}$
5	MCTS + $min(D_{NPC}) + \|max(R)\|$
6	MCTS + $min(D_{NPC}) + \frac{\|max(R)\|}{\sum D_{NPC}}$
7	MCTS + $min(D_{MOV})$
8	MCTS + $min(D_{MOV}) + \frac{1}{\sum D_{NPC}}$
9	MCTS + $min(D_{MOV}) + \|max(R)\|$
10	MCTS + $min(D_{MOV}) + \frac{\|max(R)\|}{\sum D_{NPC}}$
11	MCTS + $min(D_{MOV}) \cdot min(D_{NPC})$
12	MCTS + $min(D_{MOV}) \cdot min(D_{NPC}) + \frac{1}{\sum D_{NPC}}$
13	MCTS + $min(D_{MOV}) \cdot min(D_{NPC}) + \|max(R)\|$
14	MCTS + $min(D_{MOV}) \cdot min(D_{NPC}) + \frac{\|max(R)\|}{\sum D_{NPC}}$
15	One-Step Look Ahead
16	Random
17	Monte-Carlo Search

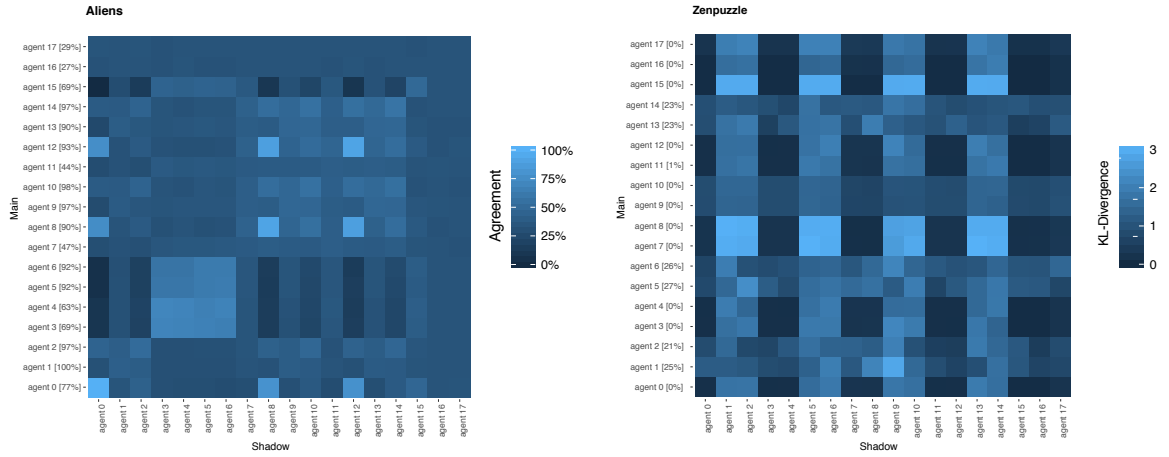
Table II summarises the agents used in the experiments and the ids assigned to them. Multiples MCTS agents using different tree policies have been tested. Figure 2 illustrates an example of agreement percentage $\mathcal{AP}$ and another of decision similarity $\mathcal{DS}$ between the main agent and the shadow agent on two tested games. An important fact to remember when looking at Figure 2a is that the probability of two random agents agreeing on the same action is $\frac{1}{|\mathcal{A}|}$. Therefore, when looking at the $\mathcal{AP}$ we should take into account and analyse what deviates from $\frac{1}{|\mathcal{A}|}$. The game Aliens is the only game where the agent has three available actions, the rest of the game is played with four available actions. The bottom-right to top-left diagonal in the matrix represents the $\mathcal{AP}$ that the agent has with itself, this particular comparison has a intrinsic meaning: it shows the coherence of the decision-making process, the higher the agreement the more consistent is the agent. This feature can be highlighted even more clearly looking at the $\mathcal{DS}$ where the complete action probability vectors are compared.

This isn't necessarily always good feature especially in competitive scenarios where a mixed strategy could be advantageous, but it's a measure of how the search process is consistent with its final decision. Picturing the action-sequence fitness landscape, a high $\mathcal{AP}$ implies that the agent shapes it in a very precise and sharp definition being able to identify consistently a path through it. In the scenarios where a lot of navigation of the level is necessary, there might be several way to reach the same end goal, this will result in the agent having a lower self-agreement.

The KL-Divergence measure adopted for $\mathcal{DS}$ hilights how distinct are the decision making processes of each agent. Using this approach we would then expect much stronger agreement along the leading diagonals of all the comparison matrices as Figure 2b. Conversely, we would also expect a much clearer distinction between agents with genuinely distinct policies.

Aliens. The game Aliens is generally easy to play, the Random agent can achieve a win rate of 27%, and the MCTS alternatives achieve win rates varied from 44% to 100%. So there are clearly some terms of the equation used in tree policy which matter more than others. The best performing agent is the agent 0 with a perfect win rate, which uses a very basic policy and chooses the action that maximises the highest value found, it's a greedy agent. An interesting pattern is observed in Figure 2a: the agents 0, 8 and 12 all share the same term $\frac{1}{\sum D_{NPC}}$ alone or together with $min(D_{MOV})$ it gives stability to the decisions taken. This is even clearer looking at the $\mathcal{DS}$ value which are respectively 0, 0.067 and 0.07 . Agent 12, the one with the best combination of $\mathcal{AP}$ and win rate, is driven by a rather peculiar policy: the first term maximises the combined minimal distance from NPCs (aliens) and movable objects (bullets), the second term minimises the sum of the distances from NPCs. This translates into a very clear and neat game-playing strategy: stay away from bullets and kill the aliens (being the fastest way to reduce $\sum D_{NPC}$). This agent is not only very strong with a 93% win rate, but also extremely fast in finding its preferred action with an average $conv = 0.26$. Even the win rate of agent 15 is not one of the best ones, the b metric highlights how an agent as 11 is intrinsically flawed. In fact, even if agent 11 constantly consumes all the budget at its disposal ($b = 1$) it gets a win rate of just 44% whilst agent 15 with a $b < 0.006$ is able to get a 69% win rate.

Brainman. This game is usually very hard for the AIs, the



(a) Aliens Pure Agreements

(b) Zenpuzzle Decision Similarities

Fig. 2: Results of two comparison scenarios between all the agents in Table II. In Figure 2a we have the comparison using the *Pure Agreement* method, the values from dark blue to light blue represent the agreement percentage (the lighter the higher). Instead in Figure 2b light blue represents very diverging action probability vectors while the darkest blue is for the case those are identical. The vertical and the horizontal dimensions of the matrix represent the main and shadow agent, respectively, in the comparison process. The main agent's win percentage is specified between square brackets in its label on the vertical axis.

best one from the batch has a win rate of 31%. Looking at the data we have noticed a high concentration of $\mathcal{AP}$ around 50% for all combination of agents from 7 to 10, this is even clearer looking at the $\mathcal{DS}$ data which is consistently below 0.2. When the policy contains the term $min(D_{MOV})$ alone, the agent is more consistent in moving far away from moving objects. Unfortunately that is exactly a behaviour that will never allow the agent to win, in fact, the key to open the door with the goal is the only movable object in the game.

Camelrace. The best way to play Camelrace is easy to understand: keep moving right until reaching the finish line. Looking into the comparison matrix $\mathcal{AP}$ for this game, we've noticed how there's a big portion of it (agents from 3 to 14) where the agents consistently agree most of the time (most values over 80%). What is interesting to highlight is how only that clustering with an $\mathcal{AP}=$ 100 (agents 8 and 7) can hit a win rate of 100% which is further highlighted by $\mathcal{DS}$ that is 0. This is due to the fact that even just few wrong actions can backfire dramatically. In fact in the game there's an NPC going straight right thus wasting few actions means risking to be overcome by it and lose the race, therefore coherence is extremely important.

Racebet2. The $\mathcal{AP}$ values for this game are harder to read, the avatar can move only in a very restricted cross-shaped area and its interaction with the game elements is completely useless until the end of the playthrough when the result of the race is obvious to the agent. This is clearly expressed by the average convergence value during the play for agent 10 shown in Figure 3. Agent 10 can not make up his mind consuming all the budget before settling for a^* ($conv = 1$), it keeps happening until the very end of the game when it has

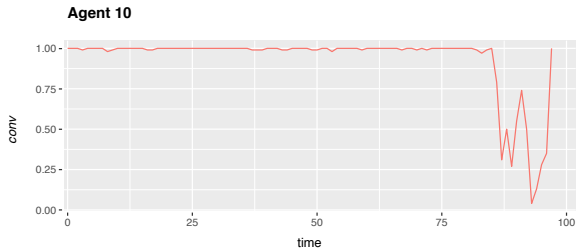


Fig. 3: The average $conv$ in the game Racebet2 for the agent 10 throughout the plays. It shows how the agent doesn't clearly have a preference over the actions until the end of the game when the value drastically drops.

a drastic drop of $conv$ meaning that the agent is now able to swiftly decide the preferred action. Potentially, an agent could stand still for most of the game and move just on the last few frames of the game. This overall irrelevance of most actions during the game is exemplified by an almost completely flat value of $\mathcal{AP}$ for most agent couples around 25%.

Zenpuzzle. This is a pure puzzle game where to win the game is not sufficient following the rewards. The $\mathcal{AP}$ values are completely flat, in this case the pure agreement doesn't provide any valuable information. However, as we can see in Figure 2b, the KL-divergence is more expressive to catch decision making differences and we can notice that generally being less consistent with itself can eventually take to perform the crucial right action to fill the whole puzzle. This is a perfect scenario to show a limit of $\mathcal{AP}$, there are several agents to win a game every four but without comparing the full action probability vector we couldn't have shown this crucial detail.

VI. Conclusion and Future Work

We have presented a set of metrics that can be used to log the decision-making process of a game-playing agent using the General Video Game AI framework. Together with these metrics, we also introduced a methodology to compare agents under the same exact conditions, both are applicable to any agent regardless of their actual implementation and the game they are meant to play. The experimental results have demonstrated how combining such methods and metrics make it possible to have a better understanding on the decision-making process of the agents. In several occasions we have seen how the measuring the agreement between a simple and not necessarily well-performing agent and the target agent, can shed some light on the implicit intentions of the latter. Such approach holds the potential for developing a set of agents with a specific well-known behaviour that can be used to analyse, using the comparison method introduced, another agent's playthrough. They could be used as an array of shadow agents, instead of a single one, and measure during the same play if and how much the behaviour of the main agent resembles that of the shadow agents. Progressively pruning the original Tree Policy we have seen how it was possible to decompose it in simple characteristic behaviours with extremely compact formulae: fleeing a type of objects, maximising the score, killing NPCs. Recognising them has been proven helpful to then understand the behaviour of more complex formulae whose behaviour is not possible to be expected a-priori.

Measuring the $conv$ has shown how it is possible to go beyond the sometimes-too-sterile win rate and to use both metrics to distinguish between more and less efficient agents. The game Zenpuzzle has clearly shown that the current set of metrics is not sufficient. The implementation of the *Shadowing Agent* and the single agents compatible with it will be released as open source code after the publication of this paper, together with the full set of comparison matrices, at www.github.com/ivanbravi/ShadowingAgentForGVGAI . In future work the metrics can be extended to represent additional information about the game states explored by the agent, such as the average events triggered, average counter for each game element just to name few as examples, but also more features from the sets envisioned in [22].

Acknowledgment

This work was funded by the EPSRC CDT in Intelligent Games and Game Intelligence (IGGI) EP/L015846/1.

References

[1] D. Perez-Liebana, J. Liu, A. Khalifa, R. D. Gaina, J. Togelius, and S. M. Lucas, "General video game ai: a multi-track framework for evaluating agents, games and content generation algorithms," *arXiv preprint arXiv:1802.10363*, 2018.

[2] M. Genesereth, N. Love, and B. Pell, "General game playing: Overview of the aaai competition," *AI magazine*, vol. 26, no. 2, p. 62, 2005.

[3] N. Love, T. Hinrichs, D. Haley, E. Schkufza, and M. Genesereth, "General game playing: Game description language specification," 2008.

[4] T. S. Nielsen, G. A. Barros, J. Togelius, and M. J. Nelson, "General video game evaluation using relative algorithm performance profiles," in *European Conference on the Applications of Evolutionary Computation*. Springer, 2015, pp. 369–380.

[5] T. Machado, I. Bravi, Z. Wang, A. Nealen, and J. Togelius, "Shopping for game mechanics," 2016.

[6] M. Ebner, J. Levine, S. M. Lucas, T. Schaul, T. Thompson, and J. Togelius, "Towards a video game description language," in *Dagstuhl Follow-Ups*, vol. 6. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2013.

[7] T. Schaul, "A video game description language for model-based or interactive learning," in *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*. IEEE, 2013, pp. 1–8.

[8] D. Perez-Liebana, S. Samothrakis, J. Togelius, T. Schaul, S. M. Lucas, A. Couëtoux, J. Lee, C.-U. Lim, and T. Thompson, "The 2014 general video game playing competition," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 8, no. 3, pp. 229–243, 2016.

[9] R. D. Gaina, A. Couetoux, D. J.N.J. Soemers, M. H.M. Winands, T. Vodopivec, F. Kirchgessner, J. Liu, S. M. Lucas, and D. Perez-Liebana, "The 2016 two-player gvgai competition," 2017.

[10] A. Khalifa, D. Perez-Liebana, S. M. Lucas, and J. Togelius, "General video game level generation," in *Proceedings of the Genetic and Evolutionary Computation Conference 2016*. ACM, 2016, pp. 253–259.

[11] A. Khalifa, M. C. Green, D. Pérez-Liébana, and J. Togelius, "General Video Game Rule Generation," in *2017 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2017.

[12] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of Monte Carlo tree search methods," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 4, no. 1, pp. 1–43, 2012.

[13] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine learning*, vol. 47, no. 2-3, pp. 235–256, 2002.

[14] M. J. Nelson, "Investigating vanilla MCTS scaling on the GVG-AI game corpus," in *Proceedings of the 2016 IEEE Conference on Computational Intelligence and Games*, 2016.

[15] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[16] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[17] D. J. Soemers, C. F. Sironi, T. Schuster, and M. H. Winands, "Enhancements for real-time monte-carlo tree search in general video game playing," in *Computational Intelligence and Games (CIG), 2016 IEEE Conference on*. IEEE, 2016, pp. 1–8.

[18] J. Méhat and T. Cazenave, "Combining uct and nested monte carlo search for single-player general game playing," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 2, no. 4, pp. 271–277, 2010.

[19] I. Bravi, "Evolving UCT alternatives for general video game playing," Master's thesis, Politechico di Milano, Italy, 2017.

[20] C. F. Sironi, J. Liu, D. Perez-Liebana, R. D. Gaina, I. Bravi, S. M. Lucas, and M. H. Winands, "Self-adaptive mcts for general video game playing," in *European Conference on the Applications of Evolutionary Computation*. Springer, 2018.

[21] P. Bontrager, A. Khalifa, A. Mendes, and J. Togelius, "Matching games and algorithms for general video game playing," in *Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2016, pp. 122–128.

[22] V. Volz, D. Ashlock, S. Colton, S. Dahlskog, J. Liu, S. M. Lucas, D. P. Liebana, and T. Thompson, "Gameplay Evaluation Measures," in *Articial and Computational Intelligence in Games: AI-Driven Game Design (Dagstuhl Seminar 17471)*, E. André, M. Cook, M. Preuß, and P. Spronck, Eds. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018, pp. 36–39.

[23] C. Browne and F. Maire, "Evolutionary game design," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 2, no. 1, pp. 1–16, 2010.

[24] J. Liu, J. Togelius, D. Perez-Liebana, and S. M. Lucas, "Evolving game skill-depth using general video game ai agents," in *2017 IEEE Congress on Evolutionary Computation (CEC)*, 2017.

[25] C. Holmgård, A. Liapis, J. Togelius, and G. N. Yannakakis, "Evolving models of player decision making: Personas versus clones," *Entertainment Computing*, vol. 16, pp. 95–104, 2016.

Q-DeckRec: A Fast Deck Recommendation System for Collectible Card Games

Zhengxing Chen
College of Comp. and Info. Sci.
Northeastern University
Boston, USA
czxttkl@gmail.com

Christopher Amato
College of Comp. and Info. Sci.
Northeastern University
Boston, USA
c.amato@northeastern.edu

Truong-Huy D. Nguyen
Dept. of Comp. and Info. Sci.
Fordham University
Bronx, USA
tnguyen88@fordham.edu

Seth Cooper
College of Comp. and Info. Sci.
Northeastern University
Boston, USA
scooper@ccs.neu.edu

Yizhou Sun
Dept. of Comp. Sci.
University of California, Los Angeles
Los Angeles, USA
yzsun@cs.ucla.edu

Magy Seif El-Nasr
College of Comp. and Info. Sci.
Northeastern University
Boston, USA
m.seifel-nasr@northeastern.edu

***Abstract*—Deck building is a crucial component in playing *Collectible Card Games* (CCGs). The goal of deck building is to choose a fixed-sized subset of cards from a large card pool, so that they work well together in-game against specific opponents. Existing methods either lack flexibility to adapt to different opponents or require large computational resources, still making them unsuitable for any real-time or large-scale application. We propose a new deck recommendation system, named Q-DeckRec, which learns a deck search policy during a training phase and uses it to solve deck building problem instances. Our experimental results demonstrate Q-DeckRec requires less computational resources to build winning-effective decks after a training phase compared to several baseline methods.**

***Index Terms*—deck recommendation, Q-learning, collectible card game**

I. Introduction

Collectible Card Games (CCGs) have been popular since the 90s, evidenced by the large player base of these kinds of games. For instance, *Magic: the Gathering* has more than 20 million players globally [1], while an online free-to-play CCG *Hearthstone* (Blizzard Inc.) reached a record of 40 million registered accounts in 2016 [2].

A CCG typically has hundreds to thousands of different cards, each of which supports specific in-game rules and effects. When playing CCGs, before each match, every player is asked to build a *deck* comprising of a subset of all available cards. While in game, each player takes turns to draw cards from their respective deck and place them on the game board to fare (e.g., attack, counter-attack, cast spell, etc.) against their opponent cards.

In general, there is no single deck which can universally win against all other decks because CCGs often design cards with sophisticated synergistic and oppositional relationships. For example, in Hearthstone, there are two distinguished types of decks that counter each other in different phases of a match. An *Aggro* deck, taking an aggressive approach, is built with cards capable of dealing damage to the opponents as quickly as possible. In contrast, a *control* deck is the opposite archetype with cards which can survive long enough to triumph in the late game through powerful but expensive cards or complex combos.

The goal of deck building is to identify a set of cards which suits the player's own play style and effectively counters either an individual opponent or a group of opponents with specific play styles and decks. As deck building is regarded as a crucial part of game play, there exist many online forums and websites for players to discuss, analyze and test deck building strategies (e.g., [3], [4]).

A deck recommendation system for the purpose of deck building can benefit players and game developers in several ways. First, it can ease choices made by players in deck building. Players may also learn new strategies of deck building and practice their skills based on recommended decks. Second, such a system can be useful to increase player's engagement, by controlling match outcomes to keep players interested [5], [6]. Deploying a deck recommendation system in certain modes (e.g., a practice mode) could help re-engage players who are frustrated with the difficulty of building effective decks. Last, from a game developer's perspective, a deck recommendation system is also useful for debugging games. For example, balancing the power of cards is an important topic in CCGs [7] or similar games [8]. The game developer can use a deck recommendation system to check whether certain combinations of cards are powerful or weak.

As a deck is a combination of cards, deck building can be formulated as a combinatorial optimization problem (COP), which relates to finding an optimal solution (the most winning-effective deck) in a finite search space of all possible decks. Deck building has a large and complex solution space. For example, the number of all possible decks in our experiment setting, which selects 15 out of 312 cards, is 1.4×10^{25}.

Previous works for deck building are mainly search algorithms, falling into two categories: heuristic searches and meta-

978-1-5386-4359-4/18/$31.00 ©2018 IEEE

heuristic searches [9]. Heuristic search methods decide which cards to include based on domain heuristics such as popularity and in-game resource curve [10]–[12]. However, heuristic methods require in-depth human knowledge and lack flexibility to adapt to different opponents. Another category is metaheuristic search, referring to high-level, problem-independent, approximate search strategies for tackling optimization problems [9]. An example is to use a *Genetic Algorithm* (GA) [13] to evolve decks towards higher winning-effectiveness through repeated modifications and selections [14], [15]. Although metaheuristic search algorithms do not require human knowledge to guide searches, they require a large computational cost *for each deck building problem instance* because: (1) the search process requires a number of evaluations of candidate solutions; (2) the evaluation of a candidate solution's quality is computationally expensive, as this requires a large number of simulated matches with complicated in-game rules.

An alternative view of solving the deck building problem is to treat it as a *sequential decision making* problem [16]. Intuitively, a deck can be built by starting in some initial card configuration (i.e., state) and applying deck modification operators (such as adding, removing, or replacing an existing card) to move to new states. The goal is to end at a final state where the deck yields a high winning chance of winning against the opponent's deck. The key challenge is to decide which operator to apply in each state. If a search policy (i.e., the mapping between states and operator choices) can be learned beforehand and simply followed while solving future problem instances, less computational resources will be needed compared to other methods requiring evaluating candidate solutions such as metaheuristic search. Such an idea is rooted in the paradigm of Reinforcement Learning (RL) algorithms concerned with learning a policy to maximize long-term rewards. In fact, leveraging RL to learn search policies for optimization problems has already been investigated in other domains, e.g., [17]–[19] but is novel for CCG deck building.

In this paper, we propose a deck recommendation system named *Q-DeckRec* whose goal is to efficiently identify winning-effective decks against specific opponents. We first model the deck building problem as solving a COP by sequential decision making, then learn a search policy by leveraging an RL algorithm on a set of "training" problem instances. The key idea is to generalize a search policy in order to find winning-effective decks and find them quickly for future problem instances. Thus, Q-DeckRec is suitable to deploy for large-scale or real-time application, e.g., an online CCG's backend to recommend winning-effective decks to a population of online players, a deck analysis website to serve hundreds of online visitors' deck building requests, or large-scale deck balancing tests.

The contributions of the paper are:

1) we formulate the deck build problem as a combinatorial optimization problem (COP);
2) we propose Q-DeckRec, an algorithm which learns a search policy for solving deck building problem instances quickly;
3) we conduct experiments to demonstrate Q-DeckRec's suitability for large-scale or real-time application. The results show that after a training phase Q-DeckRec is able to build highly winning-effective decks within 9.63 seconds of CPU time, which is not achievable by other methods.

The paper is structured as follows. Section II provides the general overview of related work. Section III describes the details of problem formulation and the proposed algorithm. Section IV and Section V describe the details of our experiments and the results, respectively. Section VI discusses our limitations and future work. Section VII concludes the study.

II. Related Work

A. Collectible Card Game Overview

Although in-game rules may vary to some extent, we focus on those CCGs similar to Hearthstone because its pattern is common and the simulator we use for our experiments is also based on it.

Each match is one-vs-one and turn-based. Each player starts with an amount of health and the goal is to destroy the opponent's health first. Each player is asked to construct a *deck* of a fixed number of cards before the actual match. During a player's turn, he plays the cards drawn from his own deck as per their rules and limited by his resource. Cards can be mainly categorized as *spells* and *minions*. Spells are played, creating an effect on the battlefield, and then are discarded. Minions, on the other hand, stay in play, and can be used to attack the enemy or other minions. There usually exist several deck archetypes in a CCG and no single deck can triumph over others universally (see the example of Aggro and Control decks in Section I).

Although a player cannot know what cards constitute the deck of his opponent before the match, he could make predictions of opponent decks and propose his deck in advance to reflect his winning philosophies. In other deck building applications, such as recommending decks in a practice mode and deck balancing tests, opponent decks can also be assumed to be known at the time of deck building.

B. Combinatorial Optimization

An optimization problem consists of an objective function and a set of problem instances. Each problem instance is defined by a set of variables and a set of constraints among those variables. A *candidate solution* to a problem instance is an assignment of values to the variables. A *feasible solution* is a candidate solution that satisfies the set of constraints. An *optimal solution* is a feasible solution that maximizes value of the objective function. A combinatorial optimization problem (COP) such as the traveling salesperson problem (TSP) is an optimization problem whose problem instances have finite numbers of candidate solutions. For many COPs, the number of candidate solutions is too large to exhaust in order to identify an optimal solution.

It is not new to approximately solve COPs through various *meta-heuristics*, i.e., high-level, problem-independent, approximate search strategies [9]. In Genetic Algorithms (GA) [13], candidate solutions evolve towards better feasible solutions iteratively with *mutation* and *crossover* operators. In each generation, the *fitness value* of every candidate solution is evaluated; the fitness value is usually the value of the objective function in the optimization problem being solved. The more fit candidate solutions are stochastically selected and modified to form a new generation. Another genre is called the *Cross-Entropy* (CE) method [20]. The central idea is that the probability of locating an optimal solution using naive random search is a rare-event probability. CE can be used to obtain a new sampling distribution so that the rare-event is more likely to occur. Sampling from the new distribution will result in near-optimal solutions.

All the metaheuristic algorithms introduced so far are *non-learning* search algorithms with one inherent disadvantage: *each time a new problem instance arises* they require a number of objective function evalutions until a sufficiently high-quality feasible solution is found. Indeed, the search process is independent between different problem instances and does not generalize a search policy which could be simply followed without objective function evaluations. If objective function evaluation is computationally expensive, non-learning search algorithms would be inefficient to solve multiple problem instances of the same COP.

Naturally, researchers are motivated to design algorithms to learn search policies for solving optimization problem instances [17], [19], [21], [22]. These algorithms lie in a broader term known as "meta-learning" [23]–[25] or "learning to learn" [26]. The learning of search policies relies on viewing the optimization process as conducting sequential decision making [16] by an optimizer agent. The optimizer agent starts in some initial state and consecutively applies operators to move to new states. The goal is to end at a final state where a high-quality feasible solution can be extracted. We call the mapping between states and operator choices as the *search policy*. If we additionally define a transition function and a reward function, we can formulate the optimization process as a *Markov Decision Process* (MDP) [27]. The optimal policy that maximizes long-term rewards can be learned or approximated by leveraging reinforcement learning (RL) algorithms [28]. The key is to properly design the MDP, especially the reward function, such that the learned policy can guide the optimizer agent quickly towards high-quality feasible solutions. For example, Zhang and Dietterich applied an RL algorithm $TD(\lambda)$ to obtain the search policy for solving NASA space shuttle scheduling problem instances [17]. Their results show that the learned search policy is more effective in the ratio of solution quality vs. CPU time than the best known non-learning search algorithm on test problem instances. Bello et. al show that a search policy parameterized as a special structure of neural network can be trained and used to solve unseen instances of TSP [19]. In Section III, we will show that under certain assumptions, a deck building problem can be formulated as a COP and a search policy learned by RL on "training" problem instances will quickly guide building winning-effective decks on future problem instances.

Besides metaheuristic and RL algorithms, problem-dependent heuristics can be used to search solutions when COPs have exploitable characteristics. However, designing heuristics is a labor intensive job. Researchers have also attempted to use supervised learning models to learn the mapping from problem instances to optimal solutions (e.g., [29]). Optimal or approximated optimal solutions need to be calculated by some solver in advance in order to provide training supervised signals. One difficulty is to design special model architectures to cope with discrete nature and constraints of COPs. For instance, in the TSP, the outputs should be constrained to sequences with no duplicated cities [29].

C. Deck Building

Ideas proposed for deck building mainly fall into two categories: heuristics and metaheuristic searches. First, some heuristic methods decide which cards to include based on the popularity of cards from historical data [10], [11]. The underlying intuition is that popularly favored cards are very likely to be strong ones. Stiegler *et al.* propose a utility system to search deck with more types of game-specific heuristics besides card popularity, including mana curve, strategic parameters, cost effectiveness and card synergies [12]. However, all heuristic methods require intensive human knowledge, lack flexibility to adapt to different opponent decks, and are not easy to transfer to other games intelligently. As far as we know, *Genetic Algorithm* (GA) [13] has been the only metaheuristic search algorithm for deck building [14], [15]. In their works, the fitness value is the average win rate of a candidate deck against a group of opponent decks while AI bots are used as a proxy for human play. However, we note in their results that a single run of GA for a particular deck building problem instance took hours or days to reach a winning-effective deck [14], [15]. This is because each fitness evaluation requires obtaining a win rate based on a number of simulated matches. The complicated in-game rules also make the simulation computationally expensive. Therefore, non-learning search algorithms like GA are not practical for any large-scale or real-time deck recommendation task.

III. Methodologies

In this section we will formally describe how the deck building problem can be cast as a COP. We will then proceed to presenting our proposed solution, *Q-DeckRec*, which solves the deck building problem from a sequential decision making perspective. In this paper, we focus on building winning-effective decks against specific individual opponents. We will leave the discussion about building decks against a group of opponents in Section VII.

A. Problem Formulation

Deck building can be formulated as a combinatorial optimization problem (COP). Suppose the goal is to build decks

as a subset of size D among a total of N cards with $N > D$ (usually N is several times larger than D). A deck can be represented as a binary vector of length N, $\boldsymbol{x} \in \mathbb{Z}_2^N$, whose components of 1's correspond to the cards included in the deck and 0's otherwise. Since a deck has a fixed size of cards, we have $\|\boldsymbol{x}\|_1 = D$. We use $\boldsymbol{x}_p$ and $\boldsymbol{x}_o$ to differentiate the deck of the player and his opponent. We use $\mathcal{A}_p$ and $\mathcal{A}_o$ to capture the play styles of the player and his opponent. $\mathcal{A}_p$ and $\mathcal{A}_o$ are play style-specific simulators that decide which cards to issue given a game instance, henceforth referred to as the *AI proxies* (artificial intelligence) of respective play styles. The evaluation function $f(\cdot)$ is defined as $f(\boldsymbol{x}_p; \boldsymbol{x}_o, \mathcal{A}_p, \mathcal{A}_o)$, which returns the winning probability of the player using $\boldsymbol{x}_p$ against the opponent using $\boldsymbol{x}_o$, with their play styles following $\mathcal{A}_p$ and $\mathcal{A}_o$ respectively. The objective function of the deck building problem is formulated as:

$$\begin{aligned} \underset{\boldsymbol{x}_p}{\arg\max} \quad & f(\boldsymbol{x}_p; \boldsymbol{x}_o, \mathcal{A}_p, \mathcal{A}_o) \\ \text{subject to} \quad & \boldsymbol{x}_p \in \mathbb{Z}_2^N, \boldsymbol{x}_o \in \mathbb{Z}_2^N, \\ & \|\boldsymbol{x}_p\|_1 = \|\boldsymbol{x}_o\|_1 = D \end{aligned} \tag{1}$$

the solution of which is denoted as $\boldsymbol{x}_p^*$.

Note that $f(\cdot)$ is a black-box function. We do not have the closed-form expression of $f(\cdot)$, but can approximate its value by simulating $\mathcal{A}_p$ and $\mathcal{A}_o$ playing against each other for a number of matches. In practice, a sufficient number of matches need to be simulated to get stable win rate estimation. Since the simulation needs to apply numerous rules of the game on each move, this is a computationally demanding operation. Each evaluation of $f(\cdot)$ was found to take non-negligible time in the order of seconds on a very powerful server machine (see Section IV). Therefore, the brute-force approach is almost infeasible to apply, as it needs evaluate an exponential number of $\boldsymbol{x}_p$ configurations, i.e., $\binom{N}{D} = N!/D!/(N-D)! = O(N^D)$. For example, in our experimental setting where $N = 312, D = 15$, it would need to exhaust around 1.4×10^{25} possibilities.

After examining potential usage scenarios, we find one assumption that could be made and exploited. Although different problem instances may use different $\mathcal{A}_p$ and $\mathcal{A}_o$, we assume that $\mathcal{A}_p$ and $\mathcal{A}_o$ come from a pool of AI proxies pre-trained by a deck recommendation system. For example, each AI proxy from the pool represents a specific play style archetype such as "aggressive" or "conservative". Under this assumption, each problem instance consists of $\boldsymbol{x}_o$ which may vary, and $\mathcal{A}_p$ and $\mathcal{A}_o$ which have been available. Therefore, there might exist deck building patterns which can be generalized. For example, if certain $\mathcal{A}_p$ is good at using Card A to counter certain $\mathcal{A}_o$, then Card A tends to appear in the optimal solution of many problem instances with the two AI proxies as the input.

In the rest of the paper, we will assume we deal with deck building problem instances of Eqn. 1 under a specific pair of $\mathcal{A}_p$ and $\mathcal{A}_o$. All the methodologies will be invariant for other pairs of AI proxies.

B. Q-DeckRec

We propose to delegate the problem of generalizing deck building patterns as a problem of generalizing a search policy in a Markov Decision Process (MDP) [27] environment, where an agent naviagates in the state space to search for the most winning-effective deck.

In the MDP, a state $s \in \mathcal{S}$ consists of a unique feasible solution $\boldsymbol{x}_p$, together with $\boldsymbol{x}_o$ and a step counter t as complement information, i.e., $s = \{\boldsymbol{x}_p, \boldsymbol{x}_o, t\}$. An action $a \in \mathcal{A}$ is defined as a card replacement to modify the current deck $\boldsymbol{x}_p$. An action replaces exactly one card in the deck $\boldsymbol{x}_p$ with another card not included currently. One special action is to keep the current deck as unmodified. Given the actions we define, the transitions between states $T : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ are always deterministic. One state applied by an action will transit to only one next state, reflecting the corresponding card modification, denoted as $\{\boldsymbol{x}_p^{(t)}, \boldsymbol{x}_o, t\}, a \to \{\boldsymbol{x}_p^{(t+1)}, \boldsymbol{x}_o, t{+}1\}$. The deck search starts from a random initial state $s_0 = \{\boldsymbol{x}_p^{(0)}, \boldsymbol{x}_o, 0\}$ and is limited to take exact D actions in one episode. We denote the states within one episode as $s_0, s_1, \cdots, s_D$. We limit the length of the horizon to be D because at most we need to replace all the cards in $\boldsymbol{x}_p^{(0)}$ to reach the optimal deck $\boldsymbol{x}_p^*$.

The problem remains as how to design the reward function $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. In the MDP, the optimal policy is the one which maximizes a defined long-term reward criterion. The key is to properly design the reward function and long-term reward criterion, such that the optimal policy is indeed the desired search policy which can lead to winning-effective decks from any state.

The long-term reward criterion defines the goal of reinforcement learning. It should encourage the optimal policy to search in the direction of winning-effective decks. We propose the following long-term reward criterion for each episode:

$$R = \sum_{t=0}^{D-1} r_t, \tag{2}$$

where r_t is the reward function over each transition. Specifically, we define r_t as the win rate between the opponent deck and the modified deck after step t with exponential amplification:

$$r_t = exp(b \cdot f(\boldsymbol{x}_p^{(t+1)}; \boldsymbol{x}_o, \mathcal{A}_p, \mathcal{A}_o)), \tag{3}$$

where b is a positive constant to adjust the extent of amplification. We choose this reward function over $r_t = f(\boldsymbol{x}_p^{(t+1)}; \boldsymbol{x}_o, \mathcal{A}_p, \mathcal{A}_o)$ in order to amplify the difference between strong and weak decks. Although the goal of deck building is to land on $s_D = \{\boldsymbol{x}_p^{(D)}, \boldsymbol{x}_o, D\}$ with $f(\boldsymbol{x}_p^{(D)}; \boldsymbol{x}_o, \mathcal{A}_p, \mathcal{A}_o)$ as high as possible, the cumulative sum of win rates provides more reward signals along the search than merely optimizing $R = r_D$. This shape of reward has also been used in previous optimization problems based on sequential decision making [22], [30]. Since we model each episode with finite horizons, we ignore the conventional reward discount factor γ in the definition of R, which is a

mathematical trick to help the convergence of RL learning in MDPs with infinite horizons.

The optimal policy can be obtained by always selecting the action with the highest optimal state-action value at each state:

$$\pi^*(s) = \arg\max_a Q^*(s,a), s = s_0, \cdots, s_{D-1}, \quad (4)$$

where $Q^*(s,a)$ is defined as the best state-action value function among all possible policies:

$$Q^\pi(s,a) = \mathbb{E}[\sum_{i=t}^{D} r_i | s_t = s, a_t = a, \pi] \quad (5)$$

$$Q^*(s,a) = \max_\pi Q^\pi(s,a) \quad (6)$$

The intuition behind $Q^*(s,a)$ is that it measures how promising applying the modification on the current deck would lead to the most winning effective deck. Following $\pi^*(s)$ would generate a series of modifications that faithfully build the optimal deck.

We propose to use a Reinforcement Learning (RL) algorithm, Q-Learning [31], to learn $Q^*(s,a)$ iteratively through observation tuples (s,a,r,s'). The simplest implementation of Q-Learning is a look-up table and a learning rate $0 < \alpha \leq 1$, with the update rule as:

$$\hat{Q}(s,a) = (1-\alpha)\hat{Q}(s,a) + \alpha(r + \max_{a'} \hat{Q}(s',a')) \quad (7)$$

Theory implies that if each action is tried in each state an infinite number of times and the magnitude of α meets certain criteria, then $\hat{Q}$ converges to Q^* [32]. However, our problem has a huge state space hence it is not possible to maintain a look-up table for all combinations of states and actions. Instead, we resort to Multi-Layer Perceptron (MLP) with parameters θ as a function approximator: $Q_\theta(s,a)$ learns to approximate the mapping of the feature representation of the state-action pair, $\mathcal{F}(s,a)$, to the optimal state-action value, $Q^*(s,a)$. More specifically, we use an MLP architecture with one input layer, one hidden layer and one output layer. Without requiring any prior domain knowledge, we simply let $\mathcal{F}(s,a) = s'$. Therefore, the input layer takes as input a state representation s', which has $2 \cdot N + 1$ dimensions. The output layer outputs a real value representing the predicted $Q^*(s,a)$. The exact specifications can be seen in Section IV. The update rule of θ is in a gradient descent fashion towards reducing so-called TD-error δ:

$$\delta := r + \max_{a'} Q_\theta(s',a') - Q_\theta(s,a) \quad (8)$$

$$\theta \leftarrow \theta + \alpha \cdot \delta \cdot \nabla Q_\theta(s,a) \quad (9)$$

To learn θ, we need to collect observation tuples (s,a,r,s') through solving "training" problem instances. Solving a training problem instance is to let Q-DeckRec take actions D times based on the current $Q_\theta(s,a)$ function in an episode. In order to generalize $Q_\theta(s,a)$ to various states, we initialize both $\boldsymbol{x}_o$ and $\boldsymbol{x}_p^{(0)}$ in s_0 randomly at the beginning of each episode. An ϵ-greedy policy is used during the training, with ϵ slowly decreasing as the learning proceeds. The policy has ϵ probability to choose non-optimal actions in the hope to escape any local optimum and discover better policies. Also, we use prioritized experience replay [33] to improve sample efficiency. Past experiences will be weighted according to the absolute value of δ. High TD-error associated experiences will be more likely to be sampled for MLP parameter learning.

The training phase of Q-DeckRec can be summarized as follows. At the beginning of each training episode, both $\boldsymbol{x}_o$ and $\boldsymbol{x}_p^{(0)}$ are randomly generated. Q-DeckRec decides how to "navigate" through states by ϵ-greedy policy and $Q_\theta(s,a)$ in D steps. All the D transitions are stored into the prioritized experience replay pool. Following that, m previous observation tuples (s,a,r,s') are sampled from the prioritized experience replay as a learning batch for updating θ as described in Eqn. 8 and 9. The loop continues after a new training episode is initiated. The training will be terminated after a time limit is reached.

After training, $Q_\theta(s,a)$ will become fixed. When solving a future problem instance, Q-DeckRec can start from s_0 with a random $\boldsymbol{x}_p^{(0)}$ and follows π^* as in Eqn. 4 in D steps. No call of $f(\cdot)$ will be needed during the search. As a comparison, non-learning search algorithms such as Genetic Algorithm require calling $f(\cdot)$ multiple times in order to evaluate fitness values for each problem instance [14], [15], while calling $f(\cdot)$ would take computational resources much heavier than calculating $Q_\theta(s,a)$. Therefore, Q-DeckRec has its superior suitability for large-scale or real-time application.

IV. Experiment Setup

To verify our method and compare with other methods, we test on an open-sourced CCG simulator *MetaStone*[1], which is based on the popular online digital CCG *Hearthstone* (Blizzard Entertainment, Inc.). All experiments run on a powerful server with Intel E5 2680 CPU's @ 2.40 GHz (56 logical CPU cores). Parallelization is implemented in three places: (1) linear algebra operations used in the MLP in Q-DeckRec; (2) match simulations evenly spread on all cores when evaluating $f(\cdot)$; (3) random deck sampling from a baseline based on Monte Carlo simulations (introduced later). Each call of $f(\cdot)$ returns a win rate based on 300 simulated matches, which on average takes 5 seconds and has around 5% standard deviation in the win rate evaluation.

We make a few decisions in setting up our experiments. We expect the experiment results can generalize under other settings. First, we use the same AI proxy to represent both $\mathcal{A}_p$ and $\mathcal{A}_o$. The AI proxy is provided by the simulator and is called *GreedyOptimizeMove*. It decides the best action by evaluating each action's consequence according to a heuristic. We do not use other AI proxies based on tree search methods because they take much longer per match. Second, we assume both players are from a specific in-game character class called

[1]https://github.com/demilich1/metastone

Warriors. The total number of available cards to Warriors is 312. Third, while in the real game certain cards can have at most one copy and all other cards can have at most two copies in the deck, we impose that every included card has two copies. This reduces our search space size for the test purpose and also follows the postulation that having two copies for every card makes the deck performance more reliable [14]. As a result, although the deck size is 30, the number of cards to be selected is 15. In summary, we have $N = 312, D = 15$ when optimizing Eqn. 1.

We set up Q-DeckRec as follows. The underlying MLP has one hidden layer and one output layer. The hidden layer consists of 1000 rectified linear units (ReLU). The output layer is a single unit which outputs a weighted sum from the activation values of the hidden layer. ϵ in the ϵ-greedy policy starts at 1 and decreases 0.0005 per training episode until it reaches 0.2. The size of a learning batch, m, is set at 64. For the prioritized experience replay [33], the exponent α is set at 0.6, the exponent β is linearly annealed from $\beta_0 = 0$ to 1 with step $1e^{-5}$. The capacity of the experience pool is 100K. The constant b in the reward function is set as 10. All the hyperparameters are chosen empirically without fine-tuning due to large computational resources required.

We compare Q-DeckRec with a Genetic Algorithm (GA), the method used in previous works for deck building [14], [15]. We implement GA with an open source library *DEAP*[2]. An individual is a candidate deck $\boldsymbol{x}_p$. The fitness value is $f(\boldsymbol{x}_p; \boldsymbol{x}_o, \mathcal{A}_p, \mathcal{A}_o)$. The mutation and crossover functions are customized to maintain the validity of individuals, similarly to what was adopted in [15]. Specifically, mutation is swapping one card in the deck with one not in the deck and crossover randomly exchanges cards not overlapped by the two decks. The population size of each generation is 10, with the mutation probability and the crossover probability both set as 0.2. Individual selection is based on a commonly used selection mechanism called *tournament* of size 3.

We also design an ad-hoc baseline which, like Q-DeckRec, requires a learning phase and does not require calling $f(\cdot)$ for solving future problem instances. The baseline conducts Monte Carlo (MC) simulations using a win rate predictor $\hat{f}(\cdot)$ to locate a solution. We first train a supervised learning model to approximate $f(\cdot)$. The training data are randomly generated pairs of decks represented as binary vectors. The labels are the evaluated win rates based on $f(\cdot)$. We choose to train an MLP with the same architecture as in Q-DeckRec. Given the same input, $\hat{f}(\cdot)$ would output faster than $f(\cdot)$ because the former does not need a real match simulation. When solving a future problem instance with opponent deck $\boldsymbol{x}_o$, we run MC simulations according to:

$$\arg\max_{\boldsymbol{x}_p \in \mathcal{X}_p} \hat{f}(\boldsymbol{x}_p, \boldsymbol{x}_o; \mathcal{A}_p, \mathcal{A}_o), \tag{10}$$

where $\mathcal{X}_p$ is a set of randomly generated decks. We denote the size of $\mathcal{X}_p$ as X. A larger X means more thorough sampling.

[2]https://github.com/DEAP/deap

In the experiments, we do not include any heuristic search method because we focus on algorithmic deck recommendation systems requiring minimal human knowledge involved. Besides GA, we do not include other metaheuristic search methods; similarly to GA, they all require calling the win rate evaluation function $f(\cdot)$ a number of times while solving each problem instance. We do not include supervised learning models which directly learns the mapping from problem instances to optimal solutions because this requires designing a specific model architecture to cope with the characteristics of the deck recommendation problem (e.g., outputs are constrained to contain K cards), which has not been studied before and requires non-trivial extra works.

Different wall time (i.e., real elapsed time) limits are imposed as the termination condition for both Q-DeckRec training and one run of GA. In this way, we can compare how long Q-DeckRec training and a GA run would take to reach similar performances. Wall time limits are chosen empirically based on observations in preliminary experiments and our limited computational resources. For GA, we try wall time limits as 10, 15, 20 and 25 minutes because performances often plateau after 20 minutes (as evidenced in the result section). Since we do not have the optimal solutions for test problem instances, we reference the solutions from 25-minute GA searches as approximated ground truths. For Q-DeckRec training, we test one, two and three days as the wall time limit. As will be shown in the result section, Q-DeckRec after three-day training can already reach the same optimality level as GA with 25 minute search. For the MC-simulation method, we use a training data set collected in three days and test $X = 67, 670, 6.7K, 67K, 670K$ and $6700K$. Note that $67K$ is around the same number Q-DeckRec calls its learned function approximator $Q_\theta(s, a)$ for solving a test problem instance[3] whereas higher values of X than $6700K$ would require too large computational resources to be practical for large-scale or real-time application.

In the rest of the paper, we will denote an algorithm as a specific approach (GA, $Q\text{-}DeckRec$, or MC) plus an associated parameter. For example, GA_{20min} and MC_{670K} are two algorithms. So are $Q\text{-}DeckRec_{1day}$ and $Q\text{-}DeckRec_{2days}$.

We generate 20 test problem instances for evaluating all algorithms. In our preliminary experiments where test problem instances are randomly generated, we often find GA only needs less than 100 calls of $f(\cdot)$ to identify decks with 100% win rate. This is because randomly generated $\boldsymbol{x}_o$ barely has any effective card synergy and can be easily beaten by a mediocre deck. In real-world applications, we believe it is more demanding to build winning-effective decks against competitive decks rather than random decks. In order to generate competitive opponent decks as test problem instances, we adopt a sequential manner as follows. We sample a deck $\boldsymbol{x}$ from the outputs of all algorithms for the last test problem

[3]As in Eqn. 4, each optimal action is decided after calculating the state-action values of all possible actions ($(N - D) \cdot D + 1$) and we need to take D actions per episode. When $N = 312$ and $D = 15$, the total number of state-action value evaluations is 66840.

TABLE I
RESULTS OF GENETIC ALGORITHM APPROACH

Search Time Wall Time / CPU Time	Func. Calls	Win Rate
10 min / 6.2 hr	110	0.61
15 min / 9.3 hr	189	0.86
20 min / 12.8 hr	267	0.93
25 min / 15.9 hr	315	0.94

TABLE II
RESULTS OF Q-DECKREC

Training Wall Time	Search Time Wall / CPU Time	Func. Calls	Win Rate
1 day		20K	0.64
2 days	0.38 sec / 9.63 sec	41K	0.88
3 days		62K	0.93

instance, where the sampling distribution is weighted by $f(\boldsymbol{x}; \boldsymbol{x}_o, \mathcal{A}_p, \mathcal{A}_o)$. We then use $\boldsymbol{x}$ as the input $\boldsymbol{x}_o$ for the next problem instance. The first test problem instance is obtained after 10 preliminary runs.

For a problem instance and an algorithm, the win rate of the returned $\boldsymbol{x}_p$ vs. the input $\boldsymbol{x}_o$ is considered as the result performance. We run each algorithm on each test problem instance 10 times. Each run is associated with a random seed, which controls the initialization of $\boldsymbol{x}_p^{(0)}$ in s_0 in Q-DeckRec, and the randomness in evolution behaviors in GA. Then, we use the median of the 10 runs as the performance for the algorithm on the problem instance. To measure the significance of the differences for each pair of algorithms, we also conduct a two-tailed paired Welch's t-tests with a confidence level 0.01 over all test instances. The null hypothesis is that the mean difference between the paired algorithms' win rates is zero.

In order to give a complete view of resource usage, we record both wall time and CPU time each algorithm takes to solve a test problem instance.

V. RESULTS AND DISCUSSION

The performances of the three kinds of methods are reported in Table I, II and III. All the reported numbers are the mean results over the 20 test problem instances. As stated, the result for each test problem instance is the median of 10 runs. Also, we find that all pairwise comparisons on the win rate are significant, *except*: (1) GA_{20min} vs. GA_{25min} (2) $Q\text{-}DeckRec_{3days}$ vs. GA_{20min} (3) $Q\text{-}DeckRec_{3days}$ vs. GA_{25min}.

TABLE III
PERFORMANCES OF MC SIMULATION APPROACH

X (Number of Samples)	Search Time Wall Time / CPU Time	Win Rate
67	0.01 sec / 0.18 sec	0.48
670	0.03 sec / 0.91 sec	0.64
$6.7K$	0.05 sec / 2.16 sec	0.75
$67K$	0.45 sec / 8.45 sec	0.84
$670K$	4.90 sec / 97.60 sec	0.82
$6700K$	36.06 sec / 1031.81 sec	0.77

First, we observe that the performances of GA and Q-DeckRec improve as the wall time limits increase in our test ranges. This meets our expectation because approximate COP solvers are supposed to get better solutions if using more computational resources. However, longer wall time limits than 20 minutes bring diminishing improvement in GA as we find there is no significant difference in the average win rate between GA_{20min} and GA_{25min}.

From Table I, we observe that GA calls the win rate evaluation function an increasing number of times as the wall time limit increases. As we stated, the win rate evaluation is computationally expensive involving simulating 300 matches. Therefore, all GA algorithms require high CPU time in the order of hours.

As shown in Table II, Q-DeckRec can solve deck building problem instances with as little computational cost as 9.63 seconds in CPU time. Meanwhile, Q-DeckRec after 3-day training can build decks as winning-effective as GA_{25min} does, as evidenced by the non-significant difference between $Q\text{-}DeckRec_{3days}$ vs. GA_{25min}. Therefore, from the CPU time perspective, Q-DeckRec is much efficient than GA (9.63 sec $\ll$ 15.9 hr) to solve a new problem instance because the computationally heavy match simulations have been "moved" to the training phase. This proves the merit of Q-DeckRec being a suitable deck recommendation system for large-scale or real-time application.

The number of function calls is $62K$ during the training of $Q\text{-}DeckRec_{3days}$. This means there are $62K$ state transitions generated from roughly $4K$ ($\approx 62K/15$) training episodes. Even if each of the $62K$ state transitions is unique, they still involve a tiny fraction of total states in our formulated state space. (The number of total states is the number of possible opponent decks times the number of possible player decks: $\binom{N}{D} \times \binom{N}{D} \approx 1.97 * 10^{50}$.) This shows that the MLP-based architecture is a well-chosen function approximator for generalizing state-action values.

For the MC-simulation method, we first report Mean Squared Error (MSE) and R^2 of the learned supervised learning model. We evaluate them using a standard 10-fold cross validation. On training data, $MSE = 0.005$ and $R^2 = 0.86$. On testing data, $MSE = 0.008$ and $R^2 = 0.79$. To our surprise, from Table III, we find that the win rate does not monotonously increase as X increases. The performance peaks at 0.84, which is significantly lower than $Q\text{-}DeckRec_{3days}$. While debugging the method, we observe that the predicted win rate (the outcome of Eqn. 10) monotonously increases as X increases. We suspect that since the supervised learning model cannot perfectly predict the real win rate, deck samples inevitably contain outlier decks with spuriously high predicted win rates. These outlier decks "trick" the MC-simulation method to select them unfortunately. The results show that the approach of building winning-effective decks in a sequential way as in Q-DeckRec is more robust.

VI. Limitations and Future Works

In order to help human players, Q-DeckRec relies on AI proxies which can accurately model players' play styles. The current used AI proxy is only based on a greedy heuristic rather than trained on human play traces. Training human-like AI proxies and integrating them to Q-DeckRec will be an important direction in our future works.

We can also improve sample efficiency in Q-DeckRec. Currently, a training episode starts with a random $s_0 = \{\boldsymbol{x}_p^{(0)}, \boldsymbol{x}_o, 0\}$. Were it generated from a card distribution learned from real matches, Q-DeckRec can focus on exploring in a smaller state space.

Next, as online CCGs often release patches to introduce new cards and modify existing cards' in-game effects, we would like to investigate how Q-DeckRec can transfer and update its knowledge without totally re-training the model [34].

Lastly, if the problem is extended to recommend winning-effective decks against a group of opponent decks $\{\boldsymbol{x}_{oi}\}_{i=1}^{k}$, there remains a question of how to design the feature representation of state-action pairs. Naive feature representations for the opponent deck group could be simply concatenating $\{\boldsymbol{x}_{oi}\}_{i=1}^{k}$. However this creates a large feature space which may not be efficient for learning. A more advanced feature representation may represent the opponent deck group in a continuous vector space, similar to word-embedding techniques from Natural Language Processing (NLP) [35]. We intend to investigate all of these in the future.

VII. Conclusions

In this paper, we propose a deck recommendation system named *Q-DeckRec*, which is able to solve deck building problem instances in large-scale and real-time after a period of training and requires minimal domain knowledge. We design experiments that demonstrate the advantages.

References

[1] G. W. Record, "Most played trading card game," http://www.guinnessworldrecords.com/world-records/most-played-trading-card-game/, Online; accessed Feb, 2018.

[2] T. Lazarides, "Hearthstone: Heroes of warcraft has more than 40 million registered players," http://toucharcade.com/2016/02/11/hearthstone-heroes-of-warcraft-has-more-than-40-million-registered-players/, Online; accessed May, 2017.

[3] HearthPwn.com, "Hearthstone Deck Builder," http://www.hearthpwn.com/deckbuilder, 2017, Online; accessed May, 2017.

[4] Icy-Veins.com, "How To Build A Deck in Hearthstone," https://www.icy-veins.com/hearthstone/how-to-build-a-deck-in-hearthstone, 2016, Online; accessed May, 2017.

[5] Z. Chen, S. Xue, J. Kolen, N. Aghdaie, K. A. Zaman, Y. Sun, and M. Seif El-Nasr, "Eomm: An engagement optimized matchmaking framework," in *Proceedings of the 26th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 2017, pp. 1143–1150.

[6] T.-H. D. Nguyen, Z. Chen, and M. S. El-Nasr, *Analytics-based AI Techniques for Better Gaming Experience*, ser. Game AI Pro, S. Rabin, Ed. Boca Raton, Florida: CRC Press, 2015, vol. 2.

[7] E. Ham, "Rarity and power: balance in collectible object games," *The International Journal of Computer Game Research*, vol. 10, no. 1, 2010.

[8] T. Mahlmann, J. Togelius, and G. N. Yannakakis, "Evolving card sets towards balancing dominion," in *Evolutionary computation (CEC), 2012 IEEE congress on*. IEEE, 2012, pp. 1–8.

[9] M. Birattari and J. Kacprzyk, *Tuning metaheuristics: a machine learning perspective*. Springer, 2009, vol. 197.

[10] F. Karsten, "Magic Math - A New Way to Determine an Aggregate Deck List," https://www.channelfireball.com/articles/magic-math-a-new-way-to-determine-an-aggregate-deck-list-rg-dragons/, Online; accessed May, 2017.

[11] W. Fancher, "Automatically Generating Magic Decks Using Deck Building Strategies," http://elvishjerricco.github.io/2015/09/24/automatically-generating-magic-decks.html, Online; accessed May, 2017.

[12] A. Stiegler, C. Messerschmidt, J. Maucher, and K. Dahal, "Hearthstone deck-construction with a utility system," in *Software, Knowledge, Information Management & Applications (SKIMA), 2016 10th International Conference on*. IEEE, 2016, pp. 21–28.

[13] J. H. Holland, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[14] P. García-Sánchez, A. Tonda, G. Squillero, A. Mora, and J. J. Merelo, "Evolutionary deckbuilding in hearthstone," in *Computational Intelligence and Games (CIG), 2016 IEEE Conference on*. IEEE, 2016, pp. 1–8.

[15] S. J. Bjørke and K. A. Fludal, "Deckbuilding in magic: The gathering using a genetic algorithm," Master's thesis, NTNU, 2017.

[16] M. L. Littman, "Algorithms for sequential decision making," 1996.

[17] W. Zhang and T. G. Dietterich, "Solving combinatorial optimization tasks by reinforcement learning: A general methodology applied to resource-constrained scheduling," *Journal of Artificial Intelligence Reseach*, vol. 1, pp. 1–38, 2000.

[18] R. Gaudel and M. Sebag, "Feature selection as a one-player game," in *International Conference on Machine Learning*, 2010, pp. 359–366.

[19] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *arXiv preprint arXiv:1611.01578*, 2016.

[20] R. Rubinstein, "The cross-entropy method for combinatorial and continuous optimization," *Methodology and computing in applied probability*, vol. 1, no. 2, pp. 127–190, 1999.

[21] K. Li and J. Malik, "Learning to optimize neural nets," *arXiv preprint arXiv:1703.00441*, 2017.

[22] Y. Chen, M. W. Hoffman, S. G. Colmenarejo, M. Denil, T. P. Lillicrap, M. Botvinick, and N. de Freitas, "Learning to learn without gradient descent by gradient descent," 2016.

[23] C. Lemke, M. Budka, and B. Gabrys, "Metalearning: a survey of trends and technologies," *Artificial intelligence review*, vol. 44, no. 1, pp. 117–130, 2015.

[24] P. Brazdil, C. G. Carrier, C. Soares, and R. Vilalta, *Metalearning: Applications to data mining*. Springer Science & Business Media, 2008.

[25] R. Vilalta and Y. Drissi, "A perspective view and survey of meta-learning," *Artificial Intelligence Review*, vol. 18, no. 2, pp. 77–95, 2002.

[26] S. Thrun and L. Pratt, *Learning to learn*. Springer Science & Business Media, 2012.

[27] R. Bellman, "A markovian decision process," *Journal of Mathematics and Mechanics*, pp. 679–684, 1957.

[28] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998, vol. 1, no. 1.

[29] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," in *Advances in Neural Information Processing Systems*, 2015, pp. 2692–2700.

[30] M. Andrychowicz, M. Denil, S. Gomez, M. W. Hoffman, D. Pfau, T. Schaul, and N. de Freitas, "Learning to learn by gradient descent by gradient descent," in *Advances in Neural Information Processing Systems*, 2016, pp. 3981–3989.

[31] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[32] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and distributed computation: numerical methods*. Prentice hall Englewood Cliffs, NJ, 1989, vol. 23.

[33] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015.

[34] M. E. Taylor and P. Stone, "Transfer learning for reinforcement learning domains: A survey," *Journal of Machine Learning Research*, vol. 10, no. Jul, pp. 1633–1685, 2009.

[35] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, 2013, pp. 3111–3119.

Inferring Design Constraints From Game Ruleset Analysis

Michael Cook and Simon Colton
The Metamakers Institute
Falmouth University
www.gamesbyangelina.org

Azalea Raad
MPI-SWS
Kaiserslautern, Germany
www.soundandcomplete.org

***Abstract*—Designing game rulesets is an important part of automated game design, and often serves as a foundation for all other parts of the game, from levels to visuals. Popular ways of understanding game rulesets include using AI agents to play the game, which can be unreliable and computationally expensive, or restricting the design space to a set of known good game concepts, which can limit innovation and creativity. In this paper we detail how ANGELINA, an automated game designer, uses an abductive analysis of game rulesets to rapidly cull its design space. We show how abduction can be used to provide an understanding of possible paths through a ruleset, reduce unplayable or undesirable rulesets without testing, and can also help discover dynamic heuristics for a game that can guide subsequent tasks like level design.**

***Index Terms*—automated game design, procedural content generation, constraint inference**

I. Introduction

Automated game design is a growing field of games research that builds on existing work in generative software, computational creativity, and general game playing. It centers around the development of software that can design complete games, either autonomously or co-creatively with the involvement of other people. Designing a complete game might include performing tasks such as inventing game mechanics; designing levels or game worlds, composing music or designing sound effects; creating visual assets or an artistic direction; or many other tasks depending on the kind of game being designed, and the kind of system being developed.

Many automated game design systems focus on rules-driven, objective-based games, where the player is presented with an objective to complete while navigating specific challenges [1]. Classic arcade games like *Space Invaders*, *Pac-Man* or *Frogger* fit into this model. These games require a clear set of rules, with win and loss conditions, and normally also require levels or starting layouts that pose specific challenge scenarios for the player. Although these rulesets are usually deterministic, they often have complex dynamics (in the sense of [2]) and their own unique set of requirements for what their levels should contain or challenge the player to do. Combined with the multiplicatively large state spaces for game rulesets and level designs, this makes automatically evaluating game designs very difficult.

Common solutions to this problem include the use of *playouts*, where an AI agent plays a game and its playtrace data is analysed to measure certain things about a game – for example, the complexity of a puzzle solution, or what score an averagely-performing agent could achieve before losing. However, playouts are computationally expensive, especially if they are used to play game designs that have obvious deficiencies in their design. Other systems use simple analytical techniques to assess game designs: for example, searching for simple rule patterns that imply an unplayable game, such as directly contradictory rules. Although this approach is effective, its use is currently restricted to very basic inferences.

In this paper we describe how we use abductive reasoning to analyse generated game rulesets, and how the resulting analysis helps us remove larger quantities of bad rulesets, without resorting to more prescriptive generation techniques. We also show that this abductive analysis of rulesets can reveal *intuitions* about a game design, which can be used to constrain level generation. We also posit how this abductive analysis might be used to incorporate subjective design constraints into automated game design.

The remainder of this paper is organised as follows: in *Background* we discuss existing approaches to solving the ruleset generation problem, and introduce the current version of the ANGELINA automated game design system; in *Static Ruleset Analysis* we describe the process of analysing game rulesets to extract possible routes through the game's logic; in *Level Constraint Inference* we extend this work to show how it can derive information that can be used to constrain the level design process; in *Design Space Analysis* we provide experimental results showing the impact these techniques have on the design space; in *Discussion and Future Work* we comment on the applicability of this technique to other kinds of game, and future work in the area; finally, in *Conclusions*, we summarise our work.

II. Background – Related Work

Automated game designers are tasked with exploring a vast state space of possible game designs. Each generative subtask – designing levels, designing game rules, designing objectives and so forth – is itself a vast possibility space, and when considered together these spaces exponentially expand and become tightly coupled to one another – a small change to a ruleset might render an entire set of levels unplayable or

978-1-5386-4359-4/18/$31.00 ©2018 IEEE

trivial. There are three broad approaches in existing automated game design work towards solving this problem.

The first approach is to leverage existing knowledge about good game design to restrict the search problem. Often this consists of building a smaller design space populated by components of existing games, and combining them in ways that preserve playability. Nelson and Mateas' work in [3] uses a 'stock set of concrete game mechanics' to build minigames. Meanwhile, The Game-O-Matic uses 'micro-rhetorics' which describe game components, and combines them using recipes which ensure the safety of the resulting game design [4]. This results in good-quality games that are reliably playable. This approach has some downsides: it is harder to discover new or surprising game designs this way, for example, as it relies on remixing to succeed. The Game-O-Matic also eschews complex level design in exchange for a simpler arrangement of shapes on the screen, which slightly reduces the complexity of the design task. Nevertheless, the Game-O-Matic demonstrates the strength of the micro-rhetoric approach, and is one of the most accomplished automated game design systems to date. We employed a similar approach in ANGELINA$_5$, which used a catalogue of known game mechanics to ensure it always produced playable rulesets.

Another common approach is to use genre-specific knowledge to focus a system on one particular type of game. In some sense, every automated game design system uses this technique, since no system to date has been truly general and cross-genre. However, systems like Data Adventures [5] or Puzzle Dice [6] focus on a very specific game structure, and as such are able to work in a reduced design space, by knowing in advance what mechanics a game might have, or being able to define objective functions for search algorithms in advance. We employed this approach in ANGELINA$_3$ to narrow our design focus on Metroidvania-style games, enabling us to define more specific objective functions for the design of levels in games the system made.

Finally, many automated game design systems use *playouts* to evaluate the games they design. Playouts involve using AI agents to play the game as a player might, and analysing the results of those playouts to infer properties about the game. Togelius and Schmidhuber use playouts to assess how hard a game is to learn, and from that infer how 'fun' the game might be as a result [7], while Khalifa *et al* use playouts to assess ruleset coverage and score achievement [8]. We have employed playouts in all versions of ANGELINA – ANGELINA$_1$, for example, used multiple playouts per game with different settings for each playout agent, to try and simulate more and less risk averse players.

III. Background – ANGELINA

ANGELINA is an automated game design project that has been in development in some form or another since 2011. The latest version of the software is developed in Unity, and currently focuses on two-dimensional, grid-based, turn-based game design. The software is designed to be run as an always-on system that is continuously developing games or discovering design knowledge. We call this *continuous automated game design* [9]. ANGELINA moves between distinct design tasks, like creating game logic or designing levels, and updates a central project file for each game it works on as tasks are completed. In this paper we focus primarily on the design of rules and levels in ANGELINA.

ANGELINA describes games using a domain specific language similar in structure to VGDL [10] or PuzzleScript [11]. Game descriptions are written in JSON, and broken down into several sections that describe different parts of the game. Figure 1 shows a template for a game, with some parts cut for length, which we will describe in stages below.

The first few lines of a game description are called the *preamble* and contain basic information about the game, including its name, the filename it is saved under, as well as basic artistic settings like the user interface colour scheme and background music. Most of these settings are optional but give the system scope to express a tone or theme through aesthetic choices. After the preamble is the *pieces* list, which defines a set of types of game object used in this game. Each piece can be instantiated one or more times in a game, and may be animate or inanimate, but these details are defined later in the game description. A piece's definition includes a name, by which it is referred to in the rules, as well as information on how to draw the object on-screen.

Most importantly for this paper, the *rules* section describes the game's core logic. It contains two lists: one containing the main game logic, which is executed at every game step; and one set of end conditions which describe ways the game can end, as well as the nature of each ending (such as a win, or a loss). A rule is comprised of two parts: a *trigger*, which is a condition that decides whether or not the rule activates; and one or more *events* which are executed in order when the trigger condition is met. We use a special syntax to allow events to reference objects 'in scope' of the rule: `$n` references the nth object mentioned in the rule so far. For example, in the rule:

```
"trigger": "OVERLAP enemy player",
"events": ["DESTROY $2"]
```

The term `$2` refers to the second object in the trigger, `player`. Thus, this rule says that when an `enemy` piece overlaps a `player` piece, the `player` piece is destroyed.

Finally, the *levels* section defines the spaces the player explores as part of the game, which is empty at the point of ruleset design. Currently, ANGELINA designs games which present a traditional sequence of levels to the player one after another. Levels are defined here in the order they are played in, and can either be loaded from another file or included as raw data. In the example in Figure 1, the level shown is written as raw data, with each number in the grid corresponding to an index into the pieces list defined earlier in the game description (this list being 1-indexed as opposed to the more traditional 0-indexing, since we use 0 to represent empty space).

```
"gamename": "Before Venturing Forth",
"filename": "before_venturing",
"music": "ominous",
"color_accent": [0.4, 0.56, 0.31],
"pieces" : [
 {
  "name": "player",
  "sprite": "fighter",
 },
 //...cut for length
],
  "rules" : [
 {
  "trigger": "OVERLAP player enemy",
  "events": ["DESTROY $2"]
 },
 {
  "trigger": "OVERLAP any wall",
  "events": ["PUSHBACK $1"]
 },
 {
  "trigger": "OVERLAP enemy player",
  "events": ["DESTROY $2"]
 },
 {
  "trigger": "PLAYER_ARROW_KEY",
  "events": ["RELMOVEALL player $1", "ENDTURN"]
 },
 {
  "trigger": "ENDTURN",
  "events": ["DO_AI_HUNT enemy player"]
 }
 //...cut for length
],
  "endconditions" : [
 {
  "outcome": "1",
  "triggers": ["ALL_COVERING player goal"],
 },
 {
  "outcome": "-1",
  "triggers": ["COUNTPIECE player 0"],
 }
],
"levels" : [
 {"type": "raw",
 "width": "5",
 "height": "5",
  "data":
     [0,4,4,0,3,
   1,0,0,0,2,
   0,0,0,0,0,
   1,0,0,0,0,
   0,0,4,0,3]
 },] //cut for length
```

Fig. 1. An abridged game description file from ANGELINA.

IV. Static Ruleset Analysis

ANGELINA's design process is broken down into distinct *tasks* which compartmentalise a certain creative act, like designing a level or testing difficulty. Game logic is currently designed in the *ruleset sketching* task, which aims to produce a set of rules which can support a game. It does not have to guarantee the ruleset will produce an exciting or interesting experience, as part of this is reliant on level design. Rather, the objective here is to identify a ruleset with the most potential, and without obvious shortcomings.

In this section we describe a process of static analysis that we use to grade and filter randomly generated rulesets, using abductive reasoning to work backwards from goal states to find paths through the design space that are initiated by player action. We show that this can identify valid solutions (and routes to failure) for a game, and even provide additional design information that can impact future phases of the game design process.

A. Simple Inspection

Before ANGELINA performs abductive analysis, it first applies a simple surface-level filtering to the ruleset being considered, similar to approaches we have used in past versions of the software. This involves searching for *a priori* logical consistencies, such as a ruleset that contains two ending conditions with the same trigger, but opposite win/loss outcomes. This example is inconsistent because, in the design space we wish to consider, the player should not be able to win and lose a game in the same time step. This simple filtering has been used in many previous versions of ANGELINA as well – in our evaluation we compare this approach in isolation to full action chain generation.s

B. Action Chain Generation

The main analysis phase uses abduction to find paths through the game's design space. The result of this analysis is a series of *action chains*. An action chain is a sequence, A, of actions, $a_1, ...a_n$, which transform a starting game state into a state which triggers an end condition. Note that these are not sequences of concrete game actions, like a solution to a puzzle or the kind of sequence of game moves that an MCTS agent might produce. Instead, each step in the action chain describes a *type* of game event, a set of one or more possible actions. For example, an event chain for an adventure game might be:

- Press arrow keys to move player
- Move player over key to collect it
- Move player over door to unlock it

This chain of events doesn't describe where the key or door is in relation to the player, or how much movement is required to get there, or whether other obstacles (like enemies) will be encountered along the way. It doesn't mention these because these can only be known by looking at a ruleset in the context of a particular level. In the absence of any levels, what this action chain expresses is that there is *some* logical sequence of actions that change the abstract state of the game from a starting state into an end state. It expresses a weak endorsement that this ruleset should yield solvable level designs. Below, we describe the algorithmic process for generating a set of action chains for a particular end condition.

When analysing a ruleset, ANGELINA will generate a set of action chains for every end condition in the ruleset.

C. Outline of Algorithm

ANGELINA begins by creating a new empty action chain, and adding a single action to it: the trigger of the end condition it is solving for. It then adds this action chain to an empty list of action chains called the *open list* – this contains every action chain that is neither complete nor abandoned. The process of generating action chains terminates when the open list is empty: that is, every action chain derived from the original end condition has either completed successfully, or been removed and ended in failure.

ANGELINA picks the first chain in the open list, and examines the last trigger in the chain. If this trigger is a player-initiated action, then the chain is considered to be complete, and it is added to a list of action chains called the *complete list*, and ANGELINA proceeds to the next chain in the open list. If this trigger is not player-initiated, then it becomes the current *goal* – ANGELINA must now find all possible ways to extend this chain by finding things that cause the goal to trigger.

To do this, ANGELINA uses a lookup table that maps triggers to *enabling rules*. For a given trigger, this table lists all rules which could, in some circumstance or other, activate the trigger. For example, the trigger `COUNTEQUAL X Y` activates when the number of objects of type `X` is equal to the integer value `Y`. It can be caused by the rules `SPAWN X`, which could increase the number from a value less than `Y` towards `Y`, or `DESTROY X`, which could decrease the active count from some value greater than `Y` towards `Y`.

For each enabling rule, ANGELINA binds the variables in the rule to the specific piece types defined in the goal trigger. For example, if our goal trigger is `COUNTEQUAL cake 2`, we search the lookup table for the unbound form of the rule, `COUNTEQUAL X Y`. When we find the enabling rules `DESTROY X` and `SPAWN X`, we rebind `X` back to `cake` from the example rule, to derive the two bound rules: `DESTROY cake` and `SPAWN cake`. We call these bound rules *subgoals*.

For each subgoal, ANGELINA now searches the game ruleset for any rule which contains the subgoal in its body. Every matching rule represents one possible way the goal trigger can be activated, and thus represents one possible way to extend the action chain. Since there may be multiple subgoals, and each subgoal may match multiple rules, we can't simply extend the current action chain we are working with. Instead, when ANGELINA finds a rule containing a subgoal, it duplicates the current action chain it is considering, adds the matched rule's trigger to the end of the action chain, and adds this new chain to the open list. When this new chain is later considered by the algorithm, this newly-added trigger will become the new goal it attempts to expand.

ANGELINA creates duplicate action chains for each rule it finds that triggers a subgoal. When it has done this for all of the subgoals, or if no subgoals are found, it removes the current action chain from the open list and discards it (since it has now either been replaced with one or more duplicates of itself that extend the chain, or found no possible extensions and is thus not a usable action chain.

D. Interpreting Action Chains

ANGELINA creates action chains for every end condition when analysing a ruleset. It can use both the contents and quantity of discovered chains to analyse a ruleset and decide whether it should be kept or discarded. The current version of ANGELINA requires a game to have at least one action chain which leads to a win condition, and one action chain which leads to a loss condition, although this is not a requirement of games in general (for example, a puzzle game like *A Good Snowman Is Hard To Build* [12] does not have a loss condition in a traditional sense).

We don't require all end conditions to be reachable, as long as the game can be both won and lost. This approach suits the current version of ANGELINA, but it does have implications for other approaches to automated game design, and future versions of this system. For example, the automatic generation of tutorials or help text is often based on an analysis of the game's ruleset. If game logic is expressed in the rules which are not used in completing the game, or if there are objectives which cannot be achieved, this might produce confusion in automatically generated help text, unless such a system also used action chains to analyse a ruleset in advance.

Another detail that may not be applicable to all systems is the requirement that all actions chains start with player action. In ANGELINA's current game engine play is turn-based and game logic is applied at regular intervals when a turn is ended. We have built the system with the assumption that a turn only ends when the player takes some kind of action. In a real-time game the game's logic is executed regardless of player input; taking action in Pac-Man is not required for the ghosts to hunt down and kill the player. Thus, for other game systems action chains may not have to start with player input, and may have secondary termination conditions (such as NPC behaviours).

V. Level Constraint Inference From Action Chains

In the previous section we described how action chains can provide insight into whether a ruleset supports valid play leading to end conditions. One of the strengths of the use of action chains is that they are agnostic to the content of the game itself. They assume nothing about the content of the levels, instead the technique looks for any possible permutation of game events that might lead to a particular end condition. However, while developing this system we realised that the action chains themselves contain contextual information that can be repurposed later in the design process, because they point to specific ways in which the game can be completed. This information not only helps us filter out rulesets – it can also be used to constrain the level design space and thus make that process more efficient as well.

By way of example: consider a (trivially simple) game with one end condition: the player wins if there is exactly one cake object in the game world. There are two rules: if the player presses an arrow key, they move a person object in the direction they press; and if a person object overlaps a cake, they destroy it. By describing the rules in plain English, the reader may have already deduced that the only valid levels for this game are ones where there are two or more cakes in the world at the beginning of the game – if there is one cake the player automatically wins, and if there are no cakes then the player cannot win, because no rule allows for us to create cakes. This constraint on the game's level design is intuitive and obvious, and it would be helpful to find a way for ANGELINA to infer this automatically.

To do this, we annotate action chains as they are generated, with constraint information that applies to that action chain only. To return to an earlier example, consider the goal trigger `COUNTEQUAL X Y`. This trigger has two enabling rules: `SPAWN X` and `DESTROY X`. In the action chain generation phase we consider both as equally valid routes through the game space, but in reality there are hidden requirements on these routes being valid: `SPAWN X` can only bring about `COUNTEQUAL X Y` if the number of `X` currently in the game is less than `Y`, and *mutatis mutandae* for `DESTROY X`. We attach these conditions to every enabling rule in the lookup table. These relationships are written by hand, and while this is not time consuming, it is a clear point of human involvement in the system. We aim to investigate automating this in the future.

Whenever ANGELINA extends an action chain using an enabling rule, it also adds any attached conditions to the action chain as well. Thus, every action chain also includes a list of constraints which must apply to a level in order for the action chain to be valid. These constraints do not affect the ruleset generation phase (currently – we discuss this later, in section VII) but they *are* used to constrain the generation of levels. Prior to the use of level constraints, ANGELINA would generate levels randomly, and then use an evolutionary system with playouts to search the level design space. Now, ANGELINA can reduce the size of the level design space before and during evolution. In the cake example above, ANGELINA can understand that there must be at least two cakes in each level design, and can filter its initial population, as well as the results of crossover or mutation, to ensure these constraints are upheld.

VI. Design Space Analysis

In the previous sections we explained that action chains help to filter and reduce the search space for both ruleset design and level design. To assess the impact our approach has on on the design spaces in question, we performed some experiments, which we describe here. As a preface to the results we give, it is worth noting that automated game design systems are highly bespoke in nature: in terms of the types of game they aim to generate; in terms of the engine they use to develop games in; in terms of the algorithms they use to generate game content and the representation they choose for each part of the game. The results we provide here are unique to ANGELINA in that sense, but we believe the scale of the results speaks to the impact of our technique, and suggests it is widely applicable to other automated game design systems.

A. Ruleset Design Space Reduction

Our first experiment aimed to assess what proportion of the generative ruleset space is filtered out by using action chains. We sampled 50,000 random rulesets generated by ANGELINA with no filtering, and then evaluated each ruleset against the simple filters described in section IV-A, and then again using the action chains described in section IV-B. Using simple filters removes 9.6% of the rulesets sampled, while using action chains and selecting only games which support at least one winning and one losing action chain removes 99.5% of the ruleset samples, a tenfold increase.

This state space reduction speaks for itself, but we can also consider it in terms of time cost also. Evaluating 50,000 rulesets with our static analysis approach took 17.3 seconds on a 2017 MacBook Pro, which filtered out over 49,500 bad rulesets without evaluating them through level design or playtesting. To contrast this, we sampled ten exploratory level design processes from ANGELINA (exploratory level design is currently its next phase after ruleset design, to evaluate a ruleset by attempting to design levels for it). On average, designing a single level for a ruleset took 4 minutes and 24 seconds. Being able to rapidly cull not just unplayable rulesets, but rulesets without proper direction or goals, saves us huge amounts of time by focusing the next phase of game design on more promising rulesets.

Removing 99.5% of rulesets may sound drastic, and may lead the reader to wonder whether the remaining rulesets are quite similar or repetitive, but in fact the remaining 0.5% is still a huge design space with a wide spectrum of good and bad games in. A common approach to content generation for games is to restrict the design language or source corpus in such a way that most of the resulting possibility space is high-quality, but derivative of the source material (for example, an earlier version of ANGELINA remixed handwritten rules in such a way that there were no unplayable combinations). Here we take the opposite approach, by giving ANGELINA a generic, high-level, parameterised design language with a vast possibility space. The advantage here is that there are many games in this space that have never been conceived of by us, and some that have never been conceived of by any game designer. But the tradeoff is a much lower ratio of good to bad games. Removing 99.5% of these rulesets is simply the first step towards tackling this problem, but the remaining 0.5% still requires a lot of filtering and observation, and will only continue to grow as we expand ANGELINA's design language in the future.

B. Level Design Space Reduction

In section V we described how action chains could also identify constraints for the level design process. To understand

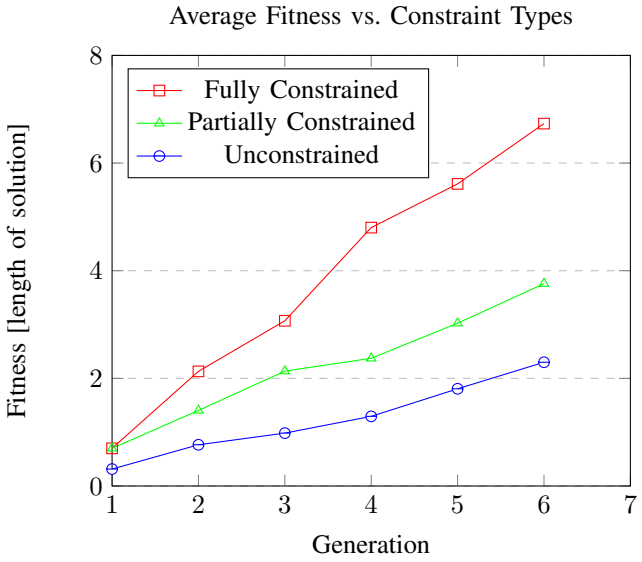


Fig. 2. A graph showing the highest fitness of the population throughout the evolutionary design of a level, sampled across 10 different level design sessions for 10 different games.

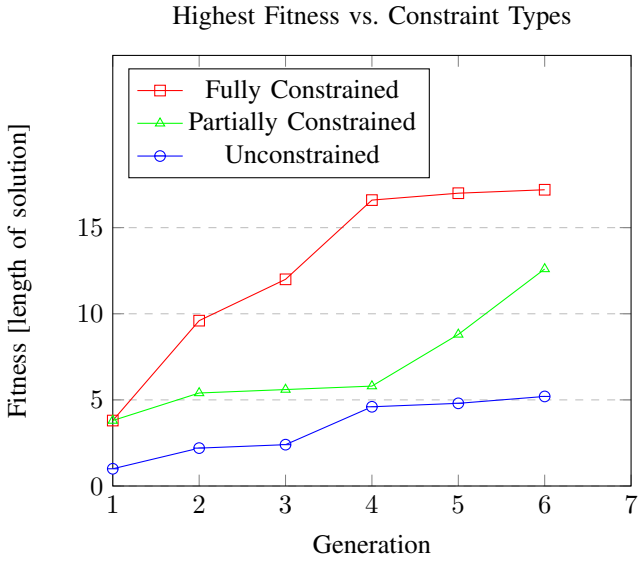


Fig. 3. A graph showing average population fitness throughout the evolutionary design of a level, sampled across 10 different level design sessions for 10 different games.

how impactful these constraints are, we assessed what proportion of the level design space these constraints typically filter out. Because every game has different constraints, we used ANGELINA to generate ten game rulesets, and then generated 50,000 levels for each game, with no constraints applied. On average, the constraints filtered out 76% of the sampled levels (ranging from 67.6% to 89.7%). By relaxing our constraints on the level generation to only require they satisfy the constraints for winning the game (for example, a tutorial level which has no lose condition) this still filters out 71% of the sampled levels on average (ranging from 55.3% to 89.7%). The range of results is derived from from how varied each ruleset is – some games may have rulesets with very specific requirements, whereas other games may rely more on the arrangement of pieces rather than their specific quantity.

ANGELINA uses an evolutionary system to design levels for its games, and an MCTS player to playtest levels. The evolutionary system uses Currently the system uses a simple objective function based on the length of the shortest solution path found by the MCTS player. This by no means relates to level or game quality in general, but we have found it to be a good initial guideline towards finding average-quality puzzle levels. We leave the question of fitness function selection for level design to a future paper. We apply level constraints both during the generation of the initial random population, and during the evolutionary process when levels are regenerated or crossed over. To assess what impact this has on the fitness and convergence of the evolutionary system, we ran some additional experiments in which ANGELINA designed a level for the same both with and without constraints.

For these experiments, we took ten game rulesets generated by ANGELINA, and generated levels for each under three different conditions: *fully constrained*, where constraints were applied to the initial population and during evolution; *partially constrained*, where constraints were applied only at the initial population generation; and *unconstrained*, where no constraints were applied at all. We measured both the highest fitness at each generation, and the average fitness of the whole population, and averaged the results for each across the ten different level design sessions. The evolutionary setup for the experiment was the same as a normal exploratory level design task in ANGELINA: a population of 12 levels, run for 5 generations, with a 5% mutation rate and elitism.

C. Results Analysis

First, we discuss Figure 2 which shows the average fitness of the whole population at each generation. We can see that both partial and full constraints initially outperform the unconstrained evolution, which is to be expected since they both start with better-filtered populations. However, in later generations the fully constrained system vastly outperforms both partial and unconstrained runs. We can clearly see that the repeated use of level constraints ensures a minimum baseline of quality in the crossover and mutation operators, which contributes to growth in the average fitness over time. By contrast, the partially constrained system tends towards the same performance as the unconstrained system, since over time it is allowing levels into its population that potentially violate the basic playability constraints.

Figure 3 shows the highest fitness scores for the three experimental setups at each generation. We can see that both partial and full constraint usage outperforms unconstrained level evolution initially, but the partial constraints then slump as it is unable to improve itself as rapidly as the fully-constrained system. We can see that later generations do see an increase in fitness, but much later than the fully-constrained

system and with a lower end fitness. We can also see evidence of the fitness plateauing for the fully constrained system around a solution length of 17, which we discuss below. Both graphs show that constraining the generation of levels in the mutation, crossover and insertion of an evolutionary system lead to higher-quality populations over time.

In the later generations of Figure 3 we see a plateau in the fully constrained system. This is not a hard ceiling – fitnesses as high as 26 have been recorded for a 4x4 grid level design – but extremely high fitnesses tend to be less reliable and more prone to collapse on repeated playouts. High fitnesses can either be because the MCTS system has truly found a stable, long solution; *or* because the MCTS playout wasn't able to properly solve the level and found a very long, imperfect solution. In the latter case, given more iterations or deeper rollouts the MCTS solver may find a drastically shorter solution, and this can even happen on repeated MCTS playouts with the same settings, since our playouts are not seeded or retained across generations. This means that the plateau represents a soft limit, beyond which it is hard to find reliably longer puzzle solutions for a 4x4 grid.

Solution length works well as a metric for ANGELINA at the current time, but its unpredictability at extremely high and low values means that in the future we hope to replace it with a more complex system that allows ANGELINA to create and select its own fitness metrics for level and game design. For now, however, solution length acts as a good guideline and has led to the creation of several games which were well-received by players – the version of ANGELINA described in this paper recently designed games live at *EGX Rezzed*, a major games expo in the UK, and had its games played by thousands of expo attendees. We're confident in using it as a guideline for level design for now, and believe these fitness graphs help demonstrate the effectiveness of design constraints on producing higher-quality levels.

VII. Discussion and Future Work

The work described in this paper represents our first steps towards rapidly culling design spaces using automated reasoning techniques. There are several aspects to this work that bear discussion in its current state, and in addition we have identified areas of future work we intend to explore.

A. Identifying Complex Contradictions

Earlier in the paper we described a simple example game where the player controls a hungry character who eats cakes by moving into them. The act of eating cakes might be expressed as the following rule:

```
"trigger": "OVERLAP player cake",
"events": [
  "DESTROY $2",
]
```

In the example earlier in the paper, the game is won when a single cake is remaining in the world. An action chain for this game can be derived as follows:

- Press arrow keys to move player
- Move player over cakes to destroy them
- Win when one cake remains

In addition, this generates a constraint that there must initially be two or more cakes in any given level. However, now suppose we adjust the rule for cake destruction as follows:

```
"trigger": "OVERLAP player cake",
"events": [
  "DESTROY $2",
  "SPAWN cake"
]
```

Now, when the player destroys a cake, another cake randomly spawns elsewhere in the game. We can easily intuit here that this game is no longer winnable: the only action in the game that reduces the number of cakes also increases it, meaning it can never be reduced or increased towards another target value. However, our current method of action chain generation doesn't take this into account. There are many other examples of subtle contradictions that are hard to detect: for example, a loss condition which is satisfied before a win condition (consider the cake-eating game, except we lose if there are exactly two cakes remaining).

We have plans to solve this problem by extending the level design constraints system that is already in place. This will allow us to express more complex constraints that can be carried up the action chain that allow us to capture notions like the fact that this action chain should result in a net decrease of the quantity of a certain object type. We are also considering exploring more formal approaches to analysing these rulesets (see *Automated Reasoning*, below).

B. Intuiting Design Knowledge

Throughout the development of ANGELINA we have consulted with game designers as well as taken part in the practice of game design ourselves. One of the interesting phenomena associated with game design is the sense of *intuition* developed by game designers that allows them to view the rules of a game, in the absence of playable content such as a level or puzzle, and infer properties the game has or simple affordances the game's ruleset offers. Designers use this when prototyping games, when considering extensions to games or trying to rebalance or fix design problems, and also use it to evaluate games designed by others.

The inferred level constraints constructed from event chains represent a very basic form of intuition about a game. They don't represent precise knowledge gained from playtesting, nor do they represent *a priori* knowledge about game design in general: they represent a way for an automated game designer to rapidly gain simple guiding principles about a particular game. In this paper we've already shown that these have value as a way of reducing the complexity of search tasks in automated game design, but from a Computational Creativity perspective this may be an impactful way for the system to demonstrate understanding and insight in game design.

We are interested in pursuing these ideas further, and seeing what other kinds of intuition-like knowledge can be gleaned from analysing rulesets prior to engaging in level design. Designers can often hypothesise about the impact of a change on a ruleset, or sketch out what affordances a particular rule might have. While this arguably involves some kind of rapid mental simulation, as well as an accumulated history of thousands of hours of playing and experiencing different kinds of game, it is nevertheless a fascinating skill that would be hugely beneficial to automated game designers. Not least because it speeds up the prototyping phase of game design, and reduces the need to exhaustively playtest ideas, but also because it provides new opportunities to frame and describe the design process to others, which is a crucial idea that underpins a lot of work in Computational Creativity [13]. Conveying intuition and hypotheses to people as a motivation for work could have a huge impact on the perception of automated game designers – not just by audiences of players, but also by people within the games industry who might be working with such software one day as co-creators [14].

C. Automated Reasoning

Our analysis process currently relies on structures such as lookup tables to bridge the semantic gap between language's keywords (like `DESTROY`) and their underlying meanings (reducing the quantity of something). ANGELINA's domain specific design language was created with readability in mind, ease of authoring, and ease of automatic modification – we didn't anticipate the need to have the language formally analysed, and so this process is not always straightforward and reasoning about more complex properties of a game gets increasingly difficult.

In the future we are considering building a model of ANGELINA's game engine that would allow ANGELINA to formally specify its games and then reason about their properties. Thus instead of looking up the fact that `DESTROY X` reduces the number of `X` objects in the world, it could reason about the actual impact that keyword has on a concrete model of a game, and use such a model to resolve complex conflicts within a ruleset or identify deeper constraints implied by the game. Work by Martens [15] and Smith [16] show different approaches to applying logical representation to game systems, which we plan to take as inspiration.

VIII. Conclusions

In this paper we describe how applying abductive logic to game rulesets can greatly reduce the size of a game design space, by filtering out rulesets that have design deficiencies, as well as providing insights into the game ruleset that can constrain the space of possible level designs. We showed that by applying action chain filtering to ANGELINA, our automated game design system, we were able to cull over 99.5% of the total ruleset possibility space, and 75% of the average total level design possibility space. This enables ANGELINA to work with much less game design knowledge supplied *a priori* in terms of game templates, and instead work through a large possibility space to identify interesting valid games, and design levels that demonstrate a better understanding of the ruleset in question.

Automated game design is a major challenge for game AI research, identified as one of the frontier problems in the field by a recent panorama paper [17]. The field has many challenges, but chief among these is the multiplicatively vast design space that results in combining generative tasks like level design or ruleset design. Developing new ways to cut down the size of these possibility spaces, in order to analyse more deeply the promising subsections of these spaces, will help the field advance further and hopefully impact neighbouring research efforts in general game playing and generative systems.

IX. Acknowledgements

This work is funded by EC FP7 grant 621403 (ERA Chair: Games Research Opportunities). Thanks to MPI-SWS for supporting the work of the first author.

References

[1] M. Cook and G. Smith, "Formalizing non-formalism: Breaking the rules of automated game design," in *Proceedings of the Foundations of Digital Games Conference*, 2015.

[2] R. Hunicke, M. Leblanc, and R. Zubek, "Mda: A formal approach to game design and game research," in *In Proceedings of the Challenges in Games AI Workshop, Nineteenth National Conference of Artificial Intelligence*, 2004.

[3] M. J. Nelson and M. Mateas, "Towards automated game design," in *AI*IA 2007: Artificial Intelligence and Human-Oriented Computing*, 2007.

[4] M. Treanor, B. Schweizer, I. Bogost, and M. Mateas, "The micro-rhetorics of game-o-matic." in *Proceedings of the Foundations of Digital Games Conference*. ACM, 2012.

[5] G. A. B. Barros, A. Liapis, and J. Togelius, "Murder mystery generation from open data," in *Proceedings of the International Conference on Computational Creativity*, 2016.

[6] C. Fernández-Vara and A. Thomson, "Procedural generation of narrative puzzles in adventure games: The puzzle-dice system," in *Proceedings of the The Third Workshop on Procedural Content Generation in Games*, 2012.

[7] J. Togelius and J. Schmidhuber, "An experiment in automatic game design," in *Proceedings of the IEEE Conference on Computational Intelligence in Games*, 2008.

[8] A. Khalifa, D. P. Liebana, S. M. Lucas, and J. Togelius, "General video game level generation," in *GECCO*. ACM, 2016, pp. 253–259.

[9] M. Cook, "A vision for continuous automated game design," in *Proceedings of the Experimental AI and Games Workshop at AIIDE*, 2017.

[10] T. Schaul, "A video game description language for model-based or interactive learning," in *Proceedings of the IEEE Conference on Computational Intelligence in Games*, 2013.

[11] S. Lavelle, "PuzzleScript," http://www.puzzlescript.net/, 2014.

[12] A. Hazelden and B. Davis, "A good snowman is hard to build," http://www.agoodsnowman.com/, 2015.

[13] J. Charnley, A. Pease, and S. Colton, "On the notion of framing in computational creativity," 2014.

[14] M. O. Riedl and A. Zook, "AI for game production," in *Proceedings of the IEEE Conference on Computational Intelligence in Games*, 2013.

[15] C. Martens, "Ceptre: A language for modeling generative interactive systems," in *AIIDE*. AAAI Press, 2015, pp. 51–57.

[16] A. M. Smith and M. Mateas, "Answer set programming for procedural content generation: A design space approach," *IEEE Trans. Comput. Intellig. and AI in Games*, vol. 3, no. 3, pp. 187–200, 2011. [Online]. Available: https://doi.org/10.1109/TCIAIG.2011.2158545

[17] G. N. Yannakakis and J. Togelius, "A panorama of artificial and computational intelligence in games," *IEEE Trans. Comput. Intellig. and AI in Games*, vol. 7, no. 4, pp. 317–335, 2015.

Intelligent Middle-Level Game Control

Amin Babadi
Department of Computer Science
Aalto University
Helsinki, Finland
amin.babadi@aalto.fi

Kourosh Naderi
Department of Computer Science
Aalto University
Helsinki, Finland
kourosh.naderi@aalto.fi

Perttu Hämäläinen
Department of Computer Science
Aalto University
Helsinki, Finland
perttu.hamalainen@aalto.fi

***Abstract*—We propose the concept of *intelligent middle-level game control*, which lies on a continuum of control abstraction levels between the following two dual opposites: 1) high-level control that translates player's simple commands into complex actions (such as pressing Space key for jumping), and 2) low-level control which simulates real-life complexities by directly manipulating, e.g., joint rotations of the character as it is done in the runner game QWOP. We posit that various novel control abstractions can be explored using recent advances in movement intelligence of game characters. We demonstrate this through design and evaluation of a novel 2-player martial arts game prototype. In this game, each player guides a simulated humanoid character by clicking and dragging body parts. This defines the cost function for an online continuous control algorithm that executes the requested movement. Our control algorithm uses Covariance Matrix Adaptation Evolution Strategy (CMA-ES) in a rolling horizon manner with custom population seeding techniques. Our playtesting data indicates that intelligent middle-level control results in producing novel and innovative gameplay without frustrating interface complexities.**

***Index Terms*—game control, physically-based simulation, multi-agent systems, online optimization, continuous control**

I. Introduction

Game control in present video games can be divided into *high-level* and *low-level* control [1]. In high-level control, the player is able to trigger actions such as punch or kick with a simple keypress. In contrast, there are low-level control approaches where the player directly manipulates the game system simulation. For example, using this approach in a martial arts game could mean that the player has to determine the torques applied to each joint of the character's body to produce a punch. The fighting game Toribash [2] and the runner game QWOP [3] are two of the games that have successfully used this kind of low-level control.

Low-level control allows maximal expressiveness, diversity/complexity, and control in game animations; it can also remove the cost of animation production in game projects. However, it usually makes the character control extremely difficult since it requires the player to manipulate several degrees of freedom with high precision, often under time pressure. On the other hand, games with high-level control usually come with a set of pre-defined smooth and natural animations. The downside is that the animations are costly to produce, and they do not allow for interesting novel movements to emerge [1].

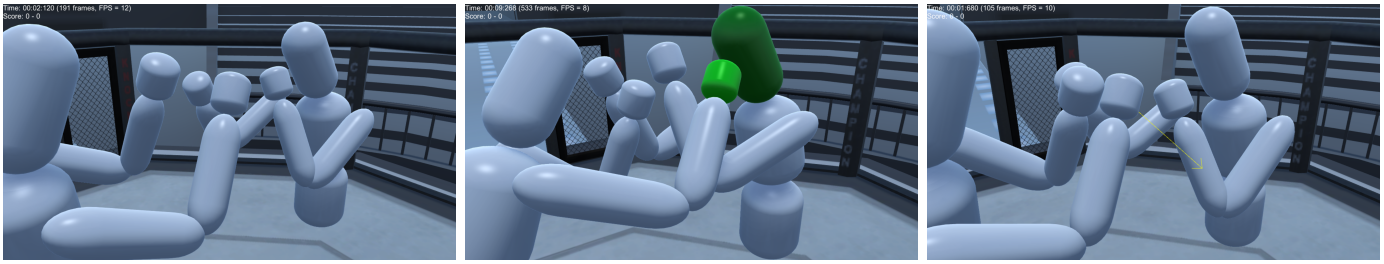
In this paper, we propose *intelligent middle-level game control* by combining usability and flexibility of high-level and low-level controls, respectively. Using this approach, the player's commands are more abstract and simple than low-level control, but more detailed and expressive than high-level control. To make this possible, we utilize recent movement artificial intelligence (AI) techniques for physically-simulated characters. To clarify this definition, consider the martial arts game example again. Suppose the player can produce the command *"use your left hand to push the opponent's right hand away"* by left-clicking on the opponent's right hand. Then, the game automatically computes and then applies the required torques for producing the requested animation, adapting to the current physical state of the characters. In other words, player commands cause the animations to be generated on the fly and no pre-recorded animations are used. The main advantages of intelligent middle-level control are as follows:

1) The synthesized movements are novel and emergent similar to low-level control, but the player can focus on more strategic planning instead of micro-managing the simulation.
2) The complexity of a simulated human body's dynamics can create interesting challenges [1]. Both novelty and complexity are desirable from the point of view of inducing a feeling of curious interest in the player [4].
3) Since movement is not limited to pre-defined animations, more expressive and precise game controls can be designed and implemented.

Middle-level control has been explored before in a few games such as Octodad [5] and the original PC version of Rag Doll Kung Fu [6], albeit with less control intelligence. The games typically use some form of inverse kinematics which limits the behaviors that can be created. We demonstrate that intelligent middle-level control with online trajectory optimization allows realistic handling of physical constraints such as joint limits and non-penetration of colliding bodies.

We believe that intelligent middle-level control has the potential for introducing various novel gameplay. To support this claim, we developed a novel 2-player martial arts game prototype in which the players are able to control their physically-based humanoid characters through giving middle-level commands. We also developed an online continuous control trajectory optimization algorithm that computes the simulation control parameters needed to produce the requested movements. Screenshots of the game are shown in Fig. 1

978-1-5386-4359-4/18/$31.00 ©2018 IEEE



(a) Task = *Null* (b) Task = *Punch in head using right hand* (c) Task = *Move right hand to specified position*

Fig. 1: Screenshots of our 2-player martial arts game prototype with 3 different tasks. The left character is controlled by the local player.

and a gameplay video is available online[1]. We evaluated this prototype by running a user study with 12 participants (6 human-vs-human pairs). The players reported that the interface enabled them to use various martial arts strategies, and that low-level controller was able to produce their commands with high precision.

The rest of the paper is organized as follows: A brief overview of literature is given in Section II. Section III explains the details of intelligent middle-level control and our martial arts game prototype. Section IV describes the details and results of the user study that was run for evaluating intelligent middle-level control in our game. Finally, Section V gives conclusions and Section VI analyzes the limitations and future lines of research in this work.

II. Related Works

In this section, we first give an overview of recent methods for synthesis and control of physically-based character animation, followed by game control interface research relevant to this work.

A. Character Movement Synthesis

Traditional animation technology has limited movement expressiveness and emergence, except for simple low-level simulation control (e.g., Toribash [2] and QWOP [3]). However, this is changing due to deep reinforcement learning and novel real-time movement optimization methods, which can endow game characters with expressive movement intelligence not limited to pre-defined animations. We refer the reader to [7] for a thorough introduction to character animation and physically-based simulation along with a survey on common techniques.

Simulation-Based Methods: In physical environments, behavior of objects and their interactions is usually difficult to model and predict. One of the most common approaches for character control in these environments is to use simulation-based methods. The basic idea behind these methods is simple: generate a number of action sequences, evaluate them using forward simulation and computing some cost function, and finally, choose the action sequence that minimizes the cost function.

If the simulation has differentiable dynamics, one can use dynamic programming version of gradient-based optimization [8] to control a variety of systems ranging from an inverted pendulum to a full humanoid. With black-box simulation, similar results were obtained by Sequential Monte Carlo sampling of control trajectories encoded as cubic splines [9]. Instead of a spline parameterization, Control Particle Belief Propagation (C-PBP) uses a Markov Random Field factorization for both sampling and smoothing trajectories [10]. Rajamäki and Hämäläinen [11] have recently shown that adding supervised learning on top of Monte Carlo tree search (MCTS) methods [12] can yield both robust control and low movement noise.

Several simulation-based methods have been developed using evolutionary computation. A recent study has used graph search along with Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [13] to develop an offline controller for solving humanoid wall climbing problems [14]. It has been shown that CMA-ES can be used in a rolling horizon manner in single-agent control problems with continuous states and actions [15] and 2-player games with discrete actions [16]. Another study has shown that performance of rolling horizon evolutionary algorithms can be improved significantly using simple population seeding techniques [17]. We are investigating this approach in the context of a more complex 2-player simulation with continuous actions and complex contact dynamics.

Reinforcement Learning: Reinforcement learning (RL) is a field of machine learning that studies how an agent should take actions in an environment in order to maximize rewards. In the past few years, RL has received a lot more attention due to remarkable results of Deep Reinforcement Learning (DRL) in Atari games [18] and the game of Go [19], [20]. These advances have inspired several breakthroughs in RL for continuous control. Mixture of actor-critic experts (MACE) accelerates learning by developing separate pairs of actors and critics such that each pair learns some part of the movement [21]. Schulman *et al.* [22] introduced a method called Trust Region Policy Optimization (TRPO) that uses a surrogate objective function and is able to learn several complex tasks such

[1] https://youtu.be/rnsSWY7HZJA

as swimming and walking. Another method called Proximal Policy Optimization (PPO) has managed to outperform TRPO by clipping the surrogate objective function [23]. The most important limitation of DRL methods is that they need a lot of simulation data and training time to learn. This can be a problem especially when iterating the reward function design.

Data-Driven Methods: Studies have shown that data-driven methods can be effective for generating robust and smooth movements. One of the studies has shown that kinematic controllers can be constructed by learning a low-dimensional space from motion capture data and interpolating in that space [24]. Motion matching is a similar kinematic method that uses a dataset of pre-recorded animations and in each frame finds the closest pose to the character's current pose such that desired future movement is produced [25]. Holden *et al.* [26] use convolutional autoencoders on a large motion capture data set to re-produce and interpolate recorded motions. Another study breaks control problem into short time fragments (0.1s in length), and learns a linear feedback control strategy for each fragment [27]. A more recent method, called DeepLoco, uses a combination of high-level and low-level controllers and is able to produce stable gaits given some reference motions [28]. Phase-functioned neural network is a recent neural network architecture that uses cyclic functions for computing the weights and is trained using a large dataset of pre-recorded animations [29]. Although data-driven methods are able to produce high-quality movements, it can be difficult to obtain the training data, and the resulting movement is limited by the data.

B. Game Control Interface

There is a lot of research on alternative input devices and interfaces, e.g., for controlling games with body movements [30]–[32]. On the other hand, some studies have proposed using traditional input devices but novel input-avatar mappings [1], [33].

A recent study has used predictive simulation for developing 6 novel game prototypes using low-level control of physically-based simulated characters [1]. Our work is close to this work as we solve the same problem of enabling expressive control of fully physically-simulated characters without excessive control difficulty. They propose predictive visualizations as the solution, whereas we explore the possibility of offloading some complexity onto the movement optimizer.

It can be argued that middle-level control is not an entirely new concept. For example, games like Octodad [5], and the original PC version of Rag Doll Kung Fu [6] have used inverse kinematics style control. Compared to Toribash [2] and QWOP [3], they provide the player with a slightly higher-level interface of directly controlling hand and feet locations instead of joint rotations. Our contribution is in demonstrating how modern physically-based optimization methods for online continuous control provide novel tools for exploring game control interfaces and abstractions.

III. Method

In this section we describe our 2-player martial arts game prototype demonstrating the intelligent middle-level control concept. At Section III-A, the details of reference model are explained. Then, at Section III-B we explain the design of intelligent middle-level control in our martial arts game prototype. At Section III-C, the structure of low-level controller is introduced. Finally, the network architecture of our martial arts game is explained in Section III-D. The values of all parameters introduced in this section are reported at Appendix.

A. Character Model

Characters in this game are upper-body humanoid characters with 16 actuated degrees of freedom (DOF) as shown in Fig. 2. Each character has 9 bones that are connected using 3-DOF ball-and-socket or 1-DOF hinge joints. We use Open Dynamics Engine (ODE) [34] for physics simulations.

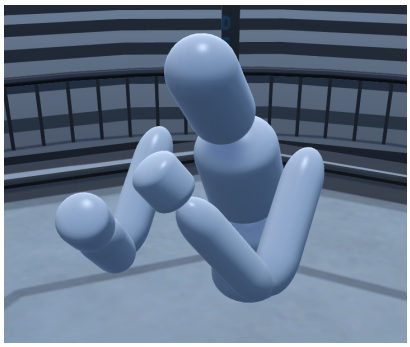


Fig. 2: Upper-body character model in its reference martial arts pose.

B. Novel Middle-Level Control for Martial Arts Games

The overall schema of our control system is shown in Fig. 3. In each frame, the current task of the character is determined and then a low-level controller plans a sequence of actions in order to complete the task. Then, the first action in the sequence is executed and the simulation goes to the next frame. This form of online optimization, i.e., 1) finding a multi-step solution, 2) applying the initial step of the solution, and 3) moving/rolling the horizon forward, is called the rolling horizon control (also known as receding horizon control) [35]. Note that the character is not necessarily given a new task in each frame in which case the task from previous frame is considered as the current task.

Tasks: Characters can have 2 main tasks: 1) *Move* hand to a specific position, and 2) *Punch* opponent either in the head or in the chest. The *Move* task is enabled using mouse drag in which case the desired position is determined based on hand's current position and mouse drag direction. A single click on opponent's head or chest enables the *Punch* task for those parts. Both tasks can be executed using either hands depending on the clicked mouse button (i.e., the left click for the left hand and the right click for the right hand). There is also a dummy *Null* task defined for specifying a character with

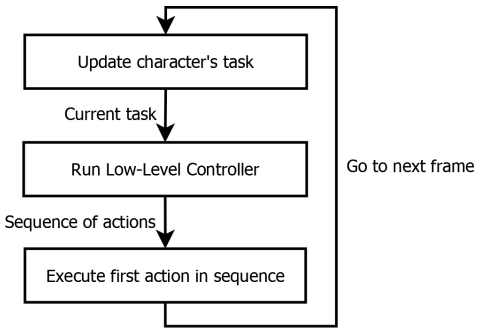


Fig. 3: Core loop in intelligent middle-level control.

no tasks. Character's task is set to *Null* if 1) current task is completed, or 2) time spent on task has passed some threshold $\tau_{Max\ Task}$.

Heads-Up Display (HUD) for tasks: The current task of character is indicated using a simple color coding as follows: 1) when the character is given a punching command, the operating hand and punching target are highlighted using green color, 2) when a successful punch is executed, the punching target will become red, and 3) when a moving command is set for one of the hands, a yellow line connects the current position of the hand to its desired position.

Additional input keys: Each player can use W/S keys to move his/her character on a horizontal line. We added this ability because in martial arts it is very important for the characters to adjust distance to their opponent. Finally, each player can rotate the camera around his/her character using A/D keys.

C. Low-Level Controller

In each frame, we need a low-level controller for achieving the character's current task. To this end, we developed an online simulation-based algorithm for physically-based continuous control. This algorithm uses CMA-ES in a rolling horizon manner along with custom population seeding techniques and outputs the best found action sequence up to a fixed horizon $\tau_{Horizon}$. CMA-ES is a common evolutionary algorithm that assumes a multi-variate Gaussian distribution as the underlying data distribution. In each iteration of CMA-ES, a new population is generated using some assumed distribution. Then, after evaluating the fitness of each individual, CMA-ES updates mean and covariance matrix of Gaussian distribution by selecting a subset of population with highest fitness values [13]. Population seeding means that in each iteration of CMA-ES, some proportion of the population is generated using external distributions. To the best of our knowledge, this is the first time that CMA-ES is used in a rolling horizon manner for online control of 3D physically-based simulated characters.

We first tried using the combined tree search and supervised learning approach of Rajamäki and Hämäläinen [11], but found that it had difficulties generating extreme dynamic movements such as punches. We then tried our present approach, as the combination of CMA-ES and a spline parameterization was successfully utilized in the dynamic climbing movement synthesis of Naderi *et al.* [14]. They however use CMA-ES for offline optimization instead of in online rolling horizon manner.

Overall Search Method: In each frame, we run CMA-ES update $n_{CMA\text{-}ES\ Updates}$ times. In each update, a population of size $n_{Population\ Size}$ is generated using mean and covariance of CMA-ES and 2 seeding techniques. Then the fitness value of each population member is computed and the CMA-ES updates its mean and covariance. After repeating this process $n_{CMA\text{-}ES\ Updates}$ times, first action in the best found action sequence is returned as the character's next action.

Spline Parameterization: We parameterize each action sequence using a cubic spline of $n_{Spline\ Points} = 3$ control points. This reduces the problem dimension significantly since we do not need to optimize each individual action in the sequence. Plus, interpolation between control points enforces coordination between body joints which results in smooth movements. We also optimize the time variable of each control point; so each spline is defined using $3 \times (16 + 1) = 51$ parameters.

Population Seeding Techniques: In each CMA-ES update, a fixed number of splines are generated using a multi-variate Gaussian distribution. The standard deviation of this distribution is σ_{Pose} and the mean is determined based on the following 2 seeding techniques: 1) $n_{Last\ Best}$ splines are generated by using the best spline found in the last frame as the mean. Note that at first CMA-ES update of each frame, we need to shift the last best spline by one frame so it becomes valid in the current frame. 2) $n_{Default\ Pose}$ splines are generated by using the default martial arts pose shown in Fig. 2 as the mean.

Fitness Computation: All action splines are evaluated in parallel using forward simulation up until some horizon $\tau_{Horizon}$ by assuming time step of $\Delta t = 1/30$ seconds and computing the reward (negative cost) in each time step. At the end of forward simulation, the fitness value of each action spline is equal to the average fitness value of all visited states during its simulation. The fitness value of a state s is computed by summing over the negation of 3 cost components as follows (the goal is to maximize the fitness):

$$Fitness\,(s) = -\left(Cost_{Pose}\,(s) + Cost_{Move}\,(s) + Cost_{Punch}\,(s)\right) \tag{1}$$

where values of cost components $Cost_X\,(s)$ are computed as follows:

1) $Cost_{Pose}\,(s)$: Cost of pose deviation is computed by finding the angle between current rotation of each bone ($q^b_{current}$) and its desired rotation ($q^b_{desired}$) as shown in the reference martial arts pose in Fig. 2. The cost is computed as:

$$Cost_{Pose}\,(s) = \sum_b \left(\frac{\angle\left(q^b_{current}, q^b_{desired}\right)}{\sigma_{Pose}} \right)^2 \tag{2}$$

where σ_{Pose} is used for indicating how much difference in rotation can be tolerated for each bone.

2) $\mathit{Cost}_{Move}(s)$: Cost of moving hand h is simply defined using the distance between current position p^h_{current} and desired position p^h_{desired} of the hand as follows:

$$\mathit{Cost}_{Move}(s) = \left(\frac{\|p^h_{\mathit{current}} - p^h_{\mathit{desired}}\|_2}{\sigma_{\mathit{Move}}}\right)^2 \quad (3)$$

where σ_{Move} is used for tuning the amount of tolerance for difference between current and desired positions.

3) $\mathit{Cost}_{\mathit{Punch}}(s)$: Punching is the most complicated cost component in our work. In a good punch, the hand touches the target with highest possible speed. Then the hand should get back to its relaxed position quickly so the character's guard is not down for a long time. In order to favor these movements, punching cost is computed as follows:

$$\begin{aligned}&\mathit{Cost}_{\mathit{Punch}}(s) = \\ &\mathbf{1}_{\mathit{punch\ not\ happened?}} \cdot \left(\frac{\|v^h_{\mathit{current}} - v^h_{\mathit{desired}}\|_2}{\sigma_{\mathit{Hand\ Velocity}}}\right)^2 - \\ &\mathbf{1}_{\mathit{punch\ happened\ now?}} \cdot \mathit{PunchPower}\left(v^h_{\mathit{current}}\right) + \\ &\mathbf{1}_{\mathit{punch\ happened\ before?}} \cdot \left(\frac{\|p^h_{\mathit{current}} - p^h_{\mathit{desired}}\|_2}{\sigma_{\mathit{Hand\ Relax\ Position}}}\right)^2\end{aligned} \quad (4)$$

where $\mathbf{1}_A$ is the indicator function and is equal to 1 if the condition A is true, and 0 otherwise. Current and desired velocity of hand are denoted by v^h_{current} and v^h_{desired}, respectively. Similarly, p^h_{current} and p^h_{desired} denote current and desired position of the hand, respectively. Similar to previous cost components, $\sigma_{\mathit{Hand\ Velocity}}$ and $\sigma_{\mathit{Hand\ Relax\ Position}}$ determine the amount of tolerance for difference between current and desired values of the hand velocity and position, respectively. The function call $\mathit{PunchPower}\left(v^h_{\mathit{current}}\right)$ maps current velocity of the hand h to a number in the range $[1000, 3000]$.

D. Network Architecture

One of the challenges for developing this prototype was how to design the network architecture. A critical limitation of previous multi-agent studies is that they mostly use competitive self-play RL, which is very slow and unreliable to train. We decided to run optimization in an interleaved manner to double our computing power. Our architecture is shown in Fig. 4. In this architecture, both the server and client do their own simulations in parallel by running the low-level controller on their devices. Then each player sends its next action to its opponent device for final simulation. Due to floating-point computation errors, the simulations on different devices are very likely to deviate. For solving this issue, server sends world state and score values to the client after its simulation is done and the client will then synchronize itself with the server.

Another important concern in this part was how partial information is handled. Each agent stores opponent's last action spline. Then, when evaluating new candidate splines, opponent's spline is simulated up until a fixed horizon $\tau_{\mathit{Opponent\ Horizon}}$ that is smaller than planning horizon, i.e., $\tau_{\mathit{Opponent\ Horizon}} < \tau_{\mathit{Horizon}}$. We did this because it is similar to real life where each character can anticipate movements of other characters by some error.

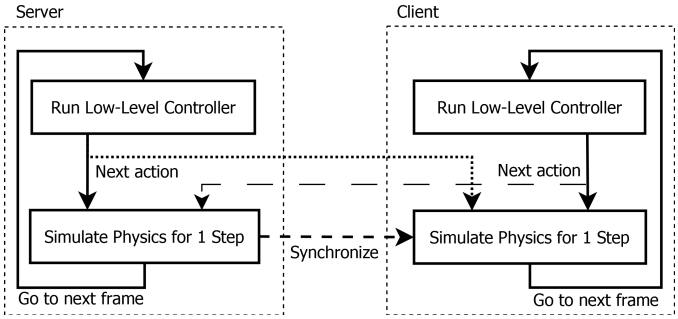


Fig. 4: Network architecture and interactions between server and client.

IV. Evaluations

A. Experimental Setup

We ran a user study involving 12 participants to assess the potentials of intelligent middle-level control in our martial arts game prototype. The participants had varying proficiency in video games and martial arts. Screenshots of our 2-player game are shown in Fig. 1 (each player controls the left character on his/her display).

Scoring: Each successful punch is rewarded with 1 to 10 points depending on its impact. The winner is the first player who gets 100 points.

Slow motion modes: The game was run in slow motion so that players have enough time for planning their movements. We hypothesized that our middle-level control interface could create a strategic *"body chess"* experience instead of a fast-paced action game. However, the camera rotates in real-time speed so the players are able to quickly adjust their point of view to see possible openings for attacks. Since we were not sure what is the right tempo for this game, we tested 3 different slow motion modes in this study. The chosen speeds for slow motion were 0.12x, 0.16x, and 0.2x. In order to find the best slow motion mode, each pair of players played one match with each slow-motion setting, with ordering of the settings counterbalanced between pairs. For the 6 pairs, each possible ordering was tested once.

Goals of User Study: The participants were asked to complete a questionnaire about the most important strengths and weaknesses of the interface. Since there are no games similar to this interface, it was not feasible to conduct a comparative quantitative evaluation. Thus, we designed the questionnaire using open-ended and qualitative questions on the following themes, with the goal of informing future work by both us and other designers and researchers:

1) What kind of combat techniques does the game allow the players to do?

2) Does the game allow novel gameplay behavior to emerge?
3) How much precision does the low-level controller have in executing players' commands?
4) Which slow motion mode is most suitable for achieving fun gameplay involving high-quality martial arts movements?
5) How can this martial arts interface be improved?

B. Results

Now we report the results obtained from 12 participants in our user study. Empty answers and answers that were irrelevant to the asked questions, are not included.

Slow Motion Mode: Slow motion modes of 0.12x, 0.16x, and 0.2x were chosen by 1, 4, and 7 people, respectively. The reported reasons are as follows:

1) The version with 0.12x speed does not produce fighting experience as the punches do not seem to be effective.
2) In the slowest mode (0.12x), it is difficult to anticipate the movements with good precision.
3) In two faster versions (0.16x and 0.2x), it is easier to control the character and react to opponent's moves.
4) The version with 0.2x speed is chosen by most of the participants because it produces the feeling of action more than other versions.

Our initial belief was that slower versions are easier to work with as they allow more time for planning and anticipation of movements. However, the results suggest that fast-paced gameplay may be more important for martial arts games.

Strategies: The main strategies reported by participants are as follows (some participants used more than one strategy):

1) Staying close and attacking aggressively (5 participants).
2) Using one hand for blocking and the other one for punching (4 participants).
3) Waiting for opponent to attack and then going for the punch (2 participants).
4) Looking for an open side and punching from that side (1 participant).
5) Getting hands through the defense of opponent (1 participant).

The variety of reported strategies and the gameplay videos suggest that middle-level control provides a good testbed for supporting different styles of gameplay.

Movement Precision: 8 participants reported that character executes the commands with high precision. Other 4 participants stated that controlling character in slow motion mode is difficult. This suggests that the control algorithm, despite its flaws, is doing a good job in synthesizing dynamic movements, but the control interface was not optimal for all participants.

Best Part: The participants were asked to name the best part of their experience while working with the interface. The answers are as follows:

1) Changing the color of body parts when punching (3 participants).
2) "*I like the idea of controlling both hands very much*".
3) "*The way that the hand and body part lit up when punching*".
4) "*The distance between characters can be adjusted in a good way*".
5) "*Seeing the game from the top*".
6) "*Fun to play against a friend*".
7) "*When it comes to punching, the character was quite creative*".
8) "*Different camera angles, realistic approach*".
9) "*Seeing nice landed punches*".
10) "*Nice to win*".

Worst Part: The participants have reported the followings as the worst parts of the interface:

1) Moving arms using mouse drag (3 participants).
2) "*Nothing was strikingly bad; but I had some trouble recollecting the right/left-click-equals-right/left-hand rule. At times, I found myself just clicking whatever clicked*".
3) "*It felt like the camera was so close to the body that you almost would like it to be first-person, and especially the camera angle above the head felt like it was from so much above it was not fun to use*".
4) "*Estimating of the time that it takes to hit*".
5) "*When trying to sweep the hands of the opponent away, it wasn't that responsive or intuitive to use*".
6) "*Both parties just end up punching each other, the game is over very quickly, and is not that fun*".
7) "*Unexpected rise of points in opponent's points when in close fight*".

Suggestions for Improvement: The participants also made the following suggestions for improving the game:

1) Adding game-like visual and audio effects (7 participants).
2) "*I think it would be cool if you could somehow with mouse make your punches' curves. Maybe dragging the mouse to show the desired curve movement for the hits*".
3) "*Moving the entire body could be possible*".
4) "*Moving hands by clicking and not dragging would make it easier to adjust hand positions*".
5) "*Allowing to crouch which makes it easier to block punches*".
6) "*Allowing to hit arms to incapacitate the other player from blocking punches using them*".

V. Conclusions

We have proposed the concept of intelligent middle-level game control, demonstrated and evaluated through a novel game prototype followed by a user study. This type of game control allows the player to produce novel gameplay through commands that are executed using a low-level controller without using any pre-recorded animations. In our 2-player martial arts game, each player controls a physically-based simulated character by giving commands such as *"punch in the chest using the left hand"*. Then a low-level controller executes the commands using a real-time control algorithm. Our online continuous control algorithm uses rolling horizon CMA-ES along with custom population seeding techniques.

We evaluated this prototype by running a user study involving 12 participants. The results show that the interface allows players to come up with various strategies for fighting. The participants have also reported that low-level controller is able to execute the commands with high precision. The users had some difficulties in mastering some of the mechanics such as camera movement and the *Move* task. However, we believe that the interface has potential for further research and novel game experiences.

VI. Limitations and Future Work

Full-Body Humanoid Characters: Our control algorithm is not currently robust enough to handle full-body humanoid characters in multi-agent environments. We have tested the algorithm successfully in locomotion tasks and single-agent settings. However, heavy perturbations make maintaining balance a very complicated problem. Since using legs is very important in martial arts, enabling this framework to work with full-body humanoid characters is a crucial direction for future work. Full-body characters should also provide enough realism to make the system useful for cognitive, strategic practicing of real martial arts, which we intend to investigate in future user studies.

Interface Design: Some reported that they prefer to see the movements in normal speed after they have given a command. However, most of the participants have stated that this character control system is fun and interesting. We are investigating possible ways for improving this interface.

Machine Learning: Currently our control algorithm does not apply any kind of machine learning for stabilizing movements. We have already got promising results by adding neural networks on top of our control algorithm in single-agent settings. Our tests show that adding machine learning can be a good approach for reducing sampling budget. Therefore, we believe that using machine learning is one of the low-hanging fruits of this work. On the other hand, being able to function without learning helps as it enables fast iteration when designing and testing the interface design.

Other Game Genres: In this work we evaluated intelligent middle-level control only in the context of martial arts games. A reasonable extension to this work is to apply this idea to other games in the sports genre where the quality of movements is important.

Acknowledgments

This work has been supported by the Academy of Finland grants 299358 and 305737.

Appendix

Table I shows the details of all important hyperparameters used in this study.

TABLE I: Algorithm parameters

Parameter	Value
Δt	$\frac{1}{30}$ s
$\tau_{Max\ Task}$	0.5 s
$\tau_{Horizon}$	0.6 s
$\tau_{Opponent\ Horizon}$	0.15 s
$n_{CMA\text{-}ES\ Updates}$	4
$n_{Population\ Size}$	16
$n_{Last\ Best}$	3
$n_{Default\ Pose}$	3
$n_{Spline\ Points}$	3
σ_{Pose}	20°
σ_{Move}	2 cm
$\sigma_{Hand\ Velocity}$	2.5 m/s
$\sigma_{Hand\ Relax\ Position}$	1 cm

References

[1] P. Hämäläinen, X. Ma, J. Takatalo, and J. Togelius, "Predictive physics simulation in game mechanics," in *Proceedings of the Annual Symposium on Computer-Human Interaction in Play*, ser. CHI PLAY '17. New York, NY, USA: ACM, 2017, pp. 497–505. [Online]. Available: http://doi.acm.org/10.1145/3116595.3116617

[2] "Toribash," http://toribash.com/, accessed: 2018-02-18.

[3] "QWOP," http://www.foddy.net/, accessed: 2018-02-18.

[4] P. J. Silvia, "Interest-the curious emotion," *Current Directions in Psychological Science*, vol. 17, no. 1, pp. 57–60, 2008.

[5] "Octodad," http://octodadgame.com/, accessed: 2018-03-09.

[6] "Rag Doll Kung Fu," http://www.ragdollkungfu.com/, accessed: 2018-03-09.

[7] T. Geijtenbeek, N. Pronost, A. Egges, and M. H. Overmars, "Interactive character animation using simulated physics." in *Eurographics (STARs)*, 2011, pp. 127–149.

[8] Y. Tassa, T. Erez, and E. Todorov, "Synthesis and stabilization of complex behaviors through online trajectory optimization," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE, 2012, pp. 4906–4913.

[9] P. Hämäläinen, S. Eriksson, E. Tanskanen, V. Kyrki, and J. Lehtinen, "Online motion synthesis using sequential monte carlo," *ACM Transactions on Graphics (TOG)*, vol. 33, no. 4, p. 51, 2014.

[10] P. Hämäläinen, J. Rajamäki, and C. K. Liu, "Online control of simulated humanoids using particle belief propagation," *ACM Transactions on Graphics (TOG)*, vol. 34, no. 4, p. 81, 2015.

[11] J. Rajamäki and P. Hämäläinen, "Augmenting sampling based controllers with machine learning," in *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. ACM, 2017, p. 11.

[12] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of monte carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in games*, vol. 4, no. 1, pp. 1–43, 2012.

[13] N. Hansen, "The cma evolution strategy: a comparing review," in *Towards a new evolutionary computation*. Springer, 2006, pp. 75–102.

[14] K. Naderi, J. Rajamäki, and P. Hämäläinen, "Discovering and synthesizing humanoid climbing movements," *ACM Transactions on Graphics (TOG)*, vol. 36, no. 4, pp. 43:1–43:11, Jul. 2017. [Online]. Available: http://doi.acm.org/10.1145/3072959.3073707

[15] S. Samothrakis, S. A. Roberts, D. Perez, and S. M. Lucas, "Rolling horizon methods for games with continuous states and actions," in *Computational Intelligence and Games (CIG), 2014 IEEE Conference on*. IEEE, 2014, pp. 1–8.

[16] J. Liu, D. Pérez-Liébana, and S. M. Lucas, "Rolling horizon coevolutionary planning for two-player video games," in *Computer Science and Electronic Engineering (CEEC), 2016 8th*. IEEE, 2016, pp. 174–179.

[17] R. D. Gaina, S. M. Lucas, and D. Pérez-Liébana, "Population seeding techniques for rolling horizon evolution in general video game playing," in *Evolutionary Computation (CEC), 2017 IEEE Congress on*. IEEE, 2017, pp. 1956–1963.

[18] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[19] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[20] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, p. 354, 2017.

[21] X. B. Peng, G. Berseth, and M. Van de Panne, "Terrain-adaptive locomotion skills using deep reinforcement learning," *ACM Transactions on Graphics (TOG)*, vol. 35, no. 4, p. 81, 2016.

[22] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International Conference on Machine Learning*, 2015, pp. 1889–1897.

[23] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[24] S. Levine, J. M. Wang, A. Haraux, Z. Popović, and V. Koltun, "Continuous character control with low-dimensional embeddings," *ACM Transactions on Graphics (TOG)*, vol. 31, no. 4, p. 28, 2012.

[25] S. Clavet, "Motion matching and the road to next-gen animation," in *Proceedings of Game Developers Conference (GDC)*, 2016.

[26] D. Holden, J. Saito, and T. Komura, "A deep learning framework for character motion synthesis and editing," *ACM Transactions on Graphics (TOG)*, vol. 35, no. 4, p. 138, 2016.

[27] L. Liu, M. V. D. Panne, and K. Yin, "Guided learning of control graphs for physics-based characters," *ACM Transactions on Graphics (TOG)*, vol. 35, no. 3, p. 29, 2016.

[28] X. B. Peng, G. Berseth, K. Yin, and M. Van De Panne, "Deeploco: dynamic locomotion skills using hierarchical deep reinforcement learning," *ACM Transactions on Graphics (TOG)*, vol. 36, no. 4, p. 41, 2017.

[29] D. Holden, T. Komura, and J. Saito, "Phase-functioned neural networks for character control," *ACM Transactions on Graphics (TOG)*, vol. 36, no. 4, p. 42, 2017.

[30] S. Krome, J. Holopainen, and S. Greuter, "Autogym: an exertion game for autonomous driving," in *Proceedings of the Annual Symposium on Computer-Human Interaction in Play*, ser. CHI PLAY '17. New York, NY, USA: ACM, 2017, pp. 33–42. [Online]. Available: http://doi.acm.org/10.1145/3116595.3116626

[31] T. Merritt, C. L. Nielsen, F. L. Jakobsen, and J. E. Grønbæk, "Glowphones: designing for proxemics play with low-resolution displays in location-based games," in *Proceedings of the Annual Symposium on Computer-Human Interaction in Play*, ser. CHI PLAY '17. New York, NY, USA: ACM, 2017, pp. 69–81. [Online]. Available: http://doi.acm.org/10.1145/3116595.3116598

[32] R. Kajastila, L. Holsti, and P. Hämäläinen, "The augmented climbing wall: high-exertion proximity interaction on a wall-sized interactive surface," in *Proceedings of the CHI Conference on Human Factors in Computing Systems*, ser. CHI '16. New York, NY, USA: ACM, 2016, pp. 758–769. [Online]. Available: http://doi.acm.org/10.1145/2858036.2858450

[33] M. Sheinin and C. Gutwin, "Quantifying individual differences, skill development, and fatigue effects in small-scale exertion interfaces," in *Proceedings of the 2015 Annual Symposium on Computer-Human Interaction in Play*. ACM, 2015, pp. 57–66.

[34] "Open dynamics engine," http://www.ode.org/, accessed: 2018-02-18.

[35] H. S. Chang, M. C. Fu, J. Hu, and S. I. Marcus, *Simulation-based algorithms for Markov decision processes*. Springer Science & Business Media, 2007.

Human-Like Playtesting with Deep Learning

Stefan Freyr Gudmundsson, Philipp Eisen, Erik Poromaa, Alex Nodet, Sami Purmonen,
Bartlomiej Kozakowski, Richard Meurling, Lele Cao
AI R&D, King Digital Entertainment, Activision Blizzard Group, Stockholm, Sweden
ai@king.com

***Abstract*—We present an approach to learn and deploy human-like playtesting in computer games based on deep learning from player data. We are able to learn and predict the most "human" action in a given position through supervised learning on a convolutional neural network. Furthermore, we show how we can use the learned network to predict key metrics of new content — most notably the difficulty of levels. Our player data and empirical data come from *Candy Crush Saga (CCS)* and Candy *Crush Soda Saga (CCSS)*. However, the method is general and well suited for many games, in particular where content creation is sequential. *CCS* and *CCSS* are non-deterministic match-3 puzzle games with multiple game modes spread over a few thousand levels, providing a diverse testbed for this technique. Compared to *Monte Carlo Tree Search (MCTS)* we show that this approach increases correlation with average level difficulty, giving more accurate predictions as well as requiring only a fraction of the computation time.**

***Index Terms*—deep learning, convolutional neural network, agent simulation, playtesting, Monte-Carlo tree search**

I. Introduction

Within the recent years, game developers have increasingly adopted a free-to-play business model for their games. This is especially true for mobile games (see e.g. [1], [2]). In the free-to-play business model, the core game is available free of charge and revenue is created through the sales of additional products and services such as additional content or in-game items. Therefore, game producers tend to continuously add content to the game to keep their users engaged and to be able to continuously monetize on a game title. For this to work out, it is important that the new content lives up to the quality expectations of the players.

The difficulty of a game has a considerable impact on a user's perceived quality. Denisova *et al.* [3] argue that challenge is the most important player experience. In trying to create the desired experience with regards to the difficulty, game designers estimate the players' skill and set game parameters accordingly. Mobile game companies usually have sophisticated tracking techniques in place to monitor how users interact with their games. This way, measures that reflect the difficulty of the game can be monitored once content has been released to players.

However, if new content would be released directly to players of the game, those would potentially be exposed to content that does not live up to their quality expectations and might abandon the game as a consequence. Therefore, game designers usually let new content be playtested and tune the parameters in an iterative manner based on data obtained from those tests before releasing the new content to players [4], [5]. Playtesting can be carried out by human test players that are given access to the new content before it is released. However, human playtesting comes at the disadvantages of introducing latency and costs in the development process. Game designers need to wait for the results from the test players before they can continue with the next iteration of their development process. Additionally, results from test players might not lead to appropriate conclusions about the general player population as the populations' skill levels can differ.

In an attempt to tackle these disadvantages several approaches for automatic playtesting have been proposed [6]–[9]. Isaksen *et al.* simulate playing levels using a simple heuristic and then analyze the level design using survival analysis. Zook *et al.* use Active Learning to automatically tune game parameters to achieve a target value in human player performance. Poromaa, similarly to Isaksen, proposes an approach, where the playtest is carried out using a Monte-Carlo Tree Search (MCTS) algorithm to simulate game play. Silva *et al.* evaluate a competitive board game by letting general (MCTS and A*) and custom AI agents play against each other.

The methods above, however, do not consider data that can be gathered from content that has been released earlier, when simulating game play. We hypothesize, that taking this data into account could lead to a game play simulation closer to human play, and therefore to better estimates of the difficulty of new content. More specifically, we built a prediction model that predicts moves from a given game state. This model is trained on moves that were executed by players on the previously released content. Once trained, the model acts as a policy, suggesting which move to execute given a game state, for an agent simulating game play. The state-of-the-art methods for predicting a move from a given state are based on Convolutional Neural Networks (CNN) [10]–[12]. CNN is a specific type of Neural Networks (NN) that is very well suited for data that comes in a grid-like structure [13]. Since the data of the problem at hand has a grid-like structure and is similar to data used in state-of-the-art research, CNN appear to be the most promising approach for the task at hand. Therefore, we rely on this approach for this research.

In our investigation, the player data is the essential part. Hence, we have to limit our research and empirical results to the games from where we can gather the required data, Candy Crush Saga (CCS) and Candy Crush Soda Saga (CCSS). It remains to be tested on other types of games. However, the approach only requires a state representation which can be

978-1-5386-4359-4/18/$31.00 ©2018 IEEE

processed by a CNN, a discrete actions space and a substantial amount of play data. The same method was used in AlphaGo [11] and the success of agents based on reinforcement learning in Atari games [14] with a similar state-action setup suggests that we can do the same for many other types of games. Interestingly, in the Atari games, the main input is the pixels of the screen so the grid-like structure can even apply to pixels. In several games, the action space can be very large, although discrete. Preprocessing of the action space might be needed without necessarily reducing the quality of the agent. For example, in bubble shooter games, e.g. Bubble Witch 3 Saga, Panda Pop, one might shrink the action space to 180 1° angles or define actions from possible destinations on the board. In linker games where the order of the links does not necessarily matter, e.g. Blossom Blast Saga, the combination of linked squares can grow exponentially with the length of the link where most combinations result in the same effect on the game and could, therefore, be defined as the same action. Bubble shooters and linker games are two types of games where we think our approach could do very well as well as clicker games, e.g. Toy Blast, Toon Blast, to mention three very popular casual game types.

In our case, the difficulty of levels is the key metric. In practice, a human-like agent can give us many more metrics from the gameplay, e.g. score distribution and a distribution of the number of moves needed to succeed. Moreover, it can become a vital part of the Quality Assurance (QA) workflow, being able to explore the relevant game space to a much larger extent than humans or random agents.

Currently, we train the agent on all the gameplay we gather. Consequently, the agent learns by averaging over all the players' policies. The policy of different players can be quite different and the result of an average policy does not have to represent the average result of different policies. The results suggest that there is, nevertheless, significant knowledge to be gained from the average policy. With player modeling or "personas" [15]–[17] we could learn policies for different clusters of players and build agents for each cluster that better predict the different policies.

A. Casual Games Genre and Match-3 Games

Casual games are a big part of the gaming industry and the genre has been growing very fast with gaming moving increasingly to mobile devices. For casual games on mobile devices it is common that the content generation is sequential, i.e. new content/levels are added to the game as the players progress. The frequency of new content can vary from every few months up to once a week. One of the biggest game types in the casual game genre is match-3 puzzle games with a few hundred million monthly active users [18].

CCS and CCSS are two versions of a match-3 game. They have a 2D board of tiles which may be left empty or filled with different items (regular and special candies) and blockers (e.g. chocolate and locked candy). A legal action is a vertical/horizontal swap of two adjacent game items that results in a vertical/horizontal match of at least 3 items of the same type or that are special candies. When included in a match, special candies remove more items from the board than just the candies that are part of the match. The empty tiles are then filled by items dropping down from above. If there are no items above an empty tile it is filled with a new random item. The diversity of this game is further enriched by different game modes, e.g. score level and timed level and behaviours of special items.

B. Contributions and Paper Organization

This paper presents an approach to estimate level difficulty in games by simulating a gameplay policy CNN learned from human gameplay. Our main contributions are:

- *a deep CNN architecture* for training agents that can play the games at hand like human players;
- *a generic framework* for estimating the level difficulty of games using agent simulations and binomial regression;
- *extensive experimental evaluations* that validate the effectiveness of our framework on match-3 games and imply practical suggestions for implementation.

In the upcoming sections, we start with related work and continue to present our proposed approach followed by thorough experimental evaluations and finally, conclusions are drawn in section VI after a short discussion about future work.

II. Related Work

Playtesting in games is used to understand the player experience and can have different perspectives, difficulty balancing and crash testing being two common examples. Player experience can be measured with various metrics [3], [19], [20]. In our context, the main focus is on playtesting as balancing the difficulty of content. To automate agent playtesting, diversified heuristic-based approaches were adopted to construct gameplay agents (e.g. [7], [21], [22]). Agents based on Monte-Carlo Tree Search, as have been proposed in [8], [23], [24], are generic and need little game-specific knowledge. Silva *et al.* [9] argued that game-specific agents usually outperform both standard MCTS and A* agents. Complying with that belief, some attempts in customizing agent heuristics began to emerge and the representative literature of that category include [25], [26], to name a few. Despite the success of hand-crafted agents on one specific game, its performance is non-transitive to other games [27] — different agents perform best in different games, which imposes difficulties when seeking to create agents effortlessly for multiple games. One of the straightforward (but inefficient) solutions is the ensemble method, so authors of [28] investigated the relative performance of 7 algorithms to formulate their approach of general game evaluation and [29] show that there is no "one-fits-all" AI-algorithm available yet in General Video Game Playing. Taken further broadly, this problem calls for a more generic form of an intelligent agent that is capable of learning the salient features embodied in different games by analyzing human-play patterns and/or directly interacting with game engines.

Although, training agents from move patterns (e.g. [30]) has been seen for over a decade, the recent advances of deep

learning techniques have moved the methodologies of this kind beyond manual feature engineering, towards a setting of end-to-end supervised learning from raw game-play data. For instance, Runarsson [31] directly approximated a policy for *Othello* game using binary classification. The works of [10] and [12] reported their CNN-based approaches achieving a prediction accuracy of 44.4% and 42% respectively on a *Go* dataset; Silver *et al.* [11] (*AlphaGo*) managed to improve the prediction accuracy to 55.7%.

In a more recent paper Silver at al. [32] managed to create an agent that could outperform any previously best artificial and human Go player in the game of Go. They proposed a novel method of Reinforcement Learning (RL) coined *AlphaGo Zero*, that was using progressive self-play without the aid of any human knowledge. In the field of multi-agent collaboration, Peng *et al.* [33] introduced a bidirectionally coordinated network with a vectorized extension of actor-critic formulation, which managed to learn several effective coordination strategies in *StarCraft*[1]. The goal of the last two approaches, however, differs from our goal in that they try to outperform, not simulate, human players. This can lead to move patterns that are different from even those of the best human players.

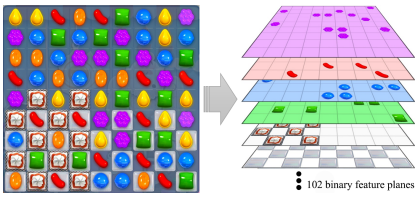


Fig. 1. An example game board of *CCS* encoded as 102-channel 2D input.

III. Approach

Our approach suggests using an agent to simulate human gameplay, creating a metric of interest. Then we relate the values of that metric observed during the simulation with the values of the same metric observed by actual, human players. As mentioned above, the metric of interest in this paper is the difficulty measured as the average success rate.

Intuitively, the more similar the strategy of the agent is to that of human players, the more should values observed during the simulation relate to the values observed from real human players. We, therefore, suggest training a CNN on human player data from previous levels to act as policy for an agent to play new, previously unseen levels.

We benchmark this approach against an approach using MCTS [34]. MCTS agents are well suited where the game environment is diverse and difficult to predict. For example, they are the state-of-the-art in General Game Playing (GGP) [35] and were a key component for the improvement of Go programs (e.g. [11], [32], [34], [36], [37]). They are search based as the agent simulates possible future states with self-play, building an asynchronous game tree in memory in the process, until it reaches the end of the search time and chooses an action to perform [30]. Our previous non-human playtesting was done with MCTS agents [8].

[1]A game published by Blizzard™: https://starcraft.com

A. CNN agent

A CNN-based agent sends the state to a policy network which gives back a probability vector over possible actions. It can be used to play greedily in each position picking the action with the highest probability in each state. Thus, playing much faster than the MCTS agent. The training of the network is done with supervised learning from player data from previous levels. Therefore, the policy network learns the most common action taken by the players in similar states.

CNNs are well suited for capturing structural correlations from data in grid-like topology [13], [38] which is often the structure of a game board, especially in casual games and match-3 games. Hence, we use a customized CNN (Fig. 2a) as our agent, which predicts the next move greedily using the current game state (i.e. board layout) as input. In this section, CCS is used as an exemplary match-3 game facilitating the explanation of the CNN agent.

B. Representation of Input and Output

The game board state, as the input of CNN, is represented as a 9×9 grid with 102 binary feature planes as demonstrated in Fig. 1. When 0-padded and stacked together, those feature planes form a 102-channel 2D input to the network. There are 4 types of input channels:

1) 80 item channels — "1" for existence of the corresponding item, "0" otherwise.
2) 20 objective channel — all "1"s when the corresponding objective (e.g. creating n striped candies) is still unfulfilled, or all "0"s.
3) 1 legal-move channel — "1" for tile that is part of a legal move, "0" otherwise.
4) 1 bias channel — a plane with "1" for every tile to allow learning a spatial bias of game board. Can be thought of as a heat map of moves.

Since the moves (output of the network) are horizontal/vertical swaps of 2 items, they are encoded as a scalar by enumerating the inner edges of the game grid (Fig. 2b), resulting in 144 possible moves.

C. Network Architecture

The architecture is selected as a result of the prestudy using both the play data from MCTS-agents and human-play data [39]. The architecture we chose consists of 11 convolutional layers. We found that a 3×3 kernel operating in stride 1 performs well for all convolutional layers. As less complex models are favored during deployment due to faster inference and training time, we empirically discovered that using only 35 filters for the first 11 convolutional layers is sufficient to maintain a relatively high accuracy. To obtain move predictions from previous convolutional layers, we followed [41]: the last convolutional layer uses exactly 144 filters (to match the numbers of possible moves) that are fed into a Global

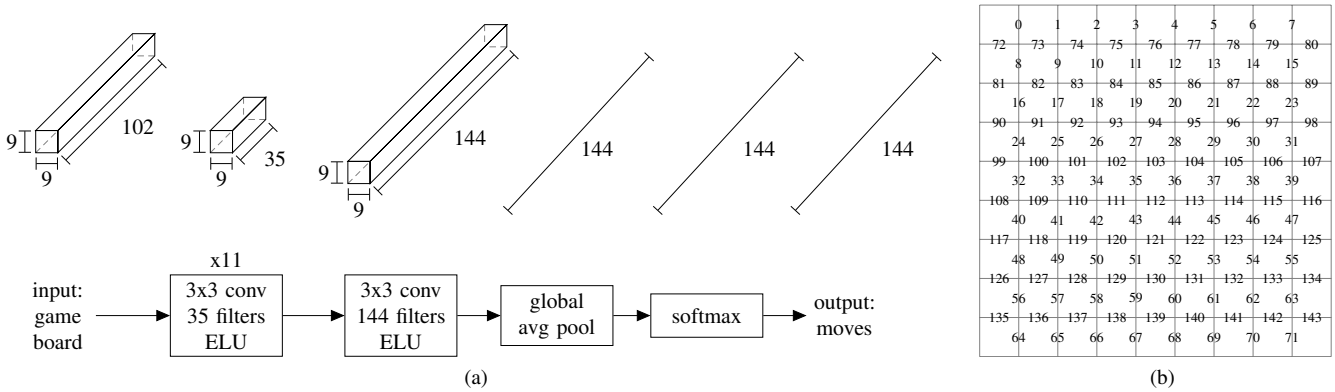


Fig. 2. Illustration of (a) the network architecture with the specification of each layer, and (b) the encoding of moves. Figures are adapted from [39], [40].

Average Pooling (GAP) layer (generating 144 scalars) right before the softmax function. It is also worth mentioning that adding the classic Fully-Connected (FC) layers with dropout regularization [38], [42] performed inferior to GAP. Despite the common application of Rectified Linear Unit (ReLU) activation function in many prominent CNN architectures (e.g. [10]–[12], [38], [42]), we opted to use the Exponential Linear Unit (ELU) function [43] instead, because it improved the validation accuracy by about 2.5%. In addition, we also experimented with batch normalization [44] and residual network [45] but neither managed to provide better generalization capability.

D. Prediction Models

The strength of an agent lies in its prediction ability, i.e. how accurately it can predict the difficulty of new content — a new level. To compare our agents, we must, therefore, build prediction models based on historical data and compare the prediction performance on new content. We measure the difficulty as the success rate, calculated as the ratio between the total number of successes and the total number of attempts. For CCS and CCSS we use data from 800 levels to build the prediction model. Then we predict the difficulty of succeeding 200 levels that have not been revealed to the training of the CNN policy network. Gathering the data for the MCTS attempts was the limiting factor where time allowed for running on 1,000 levels. We try to build the best possible prediction model for each agent. The results are therefore inevitably subject to human choices in the prediction modeling. However, we do not think this biases the comparison. The prediction models are based on binomial regression using level type features and the agents success rates as inputs. The model type and input features should not favor the CNN agents over the MCTS agent.

We use three different measures to compare the predictive power: 1) mean absolute error (MAE) between the estimated success rate and the actual success rate, 2) the percentage of test points lying outside the 95% prediction bands, and 3) standard deviation (STDDEV) of random effects. Measuring from the perspective of generalization capability, we are in favour of predictions for new levels with as little error or bias as possible. The anticipated prediction error can be expressed through the STDDEV of random effects and MAE and the bias is indicated by the percentage of points outside of the 95% prediction bands.

E. Binomial Regression

Prior to building a statistical model that expresses the players' success rate ρ_{player} using agent success rate ρ_{agent}, we observe that they do not need to linearly map to each other. For the following reasons:

- The agent and players performance depends in a different way on the game mode and features present on the board
- Players show higher success rate in the presence of game features requiring deeper strategic thinking
- The agent is much less random than players. It is because (a) the agent is a single player while human-players belong to a large group of millions of individuals playing with different skills and strategies; (b) agents follow their own policy to the point and that leads to highly correlated results.
- The average success rate observed for players is limited in its value. The same does not hold for a single player or a single agent. The observed relationship between the agent and players cannot hold for levels where the agent needs much more attempts to succeed than the average observed for the population
- We have observed that the agents and players exhibit different sensitivity to increased difficulty. The difference does not need to be linear.

In addition to limitations explained above, the model chosen needs to support rate values — ranging from 0 (when the agent fails on all attempts) to 1 (when the agent succeeds on all attempts) and the prediction, including the prediction uncertainty, needs to stay within this range of values. For that reason, we model the relationship $\rho_{\text{player}} \sim \rho_{\text{agent}}$ with binomial regression.

To account for the difference in difficulty in the presence of different board elements we add features available on the

board as covariates so that the model becomes:

$$\text{logit}(\rho_{\text{player},i}) = \beta_0 + \beta_1 \cdot f(\rho_{\text{agent},i}) + \sum_j \beta_j \cdot x_j^{<i>} + \epsilon_i \ , \quad (1)$$

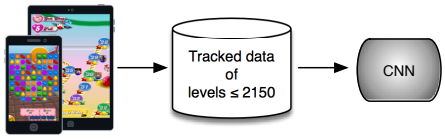
where f denotes a transformation of $\rho_{\text{agent},i}$ making it linear in logit scale; i is the index of observations; $x_j^{<i>}$ denotes the j-th feature for the i-th observation; and ϵ is the error term. The data we model is aggregated per level, i.e. each data point is represented by the average success rate for players and the average success rate obtained for the agents. The problem with this approach is that binomial regression imposes a certain limit on uncertainty. The expected variance of the observed success rate is formulated as $\text{Var}(\rho) = \rho(1-\rho)/n$, where n represents the number of attempts. For both ends of the success rate range (i.e. $\rho = 0$ or $\rho = 1$), the expected variance is 0 and for the middle point (i.e. $\rho = 0.5$) it takes its maximum value and becomes $0.25/n$. The dispersion of data collected exceeds the limits imposed by the binomial model. This phenomenon is known as overdispersion [46] and if not taken into account it causes problems with inference. In particular it leads to underestimation of the uncertainty around the estimated parameters and as a consequence to biased predictions.

The common strategies to account for overdispersion include adding new features and transforming the existing features, neither of which solves our problem since we know that the overdispersion is caused by the agent behavior which tends to be self-correlated. In statistical literature, such a situation is described as clustered measurements [46]. In our case, a cluster is a game level for which we observe an average success rate. Such data is assumed to be affected by two random processes: (a) within-cluster randomness (uncertainty of the measurement of ρ for a single game level) that is already captured by the term ϵ in equation (1); and (b) between-cluster randomness (uncertainty resulting from data being correlated), which we model by introducing term κ to (1) and hence obtain the improved model:

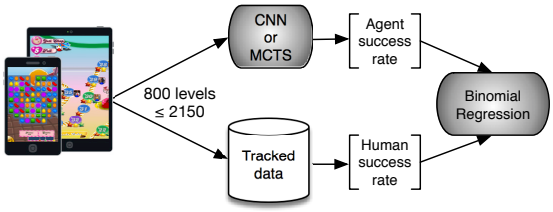
$$\text{logit}(\rho_{\text{player},i}) = \beta_0 + \beta_1 \cdot f(\rho_{\text{agent},i}) + \sum_j \beta_j \cdot x_j^{<i>} + \epsilon_i + \kappa_i \ , \quad (2)$$

where both random terms are expected to be normally distributed around zero. This kind of model belongs to a family of generalized linear mixed models. Here, κ_i is a random effect and all other features are fixed effects. The model estimates all parameters (i.e. β_0, β_1 and β_j) together with the variance of κ. The prediction of our model has two parts: the deterministic part expressed by the logit model and the random part expressed by κ. Since there is no closed-form solution for obtaining prediction bands resulting from (2), we obtain them instead by simulation and bootstrapping. The final result of the modeling is shown in Fig. 4 and Table II.

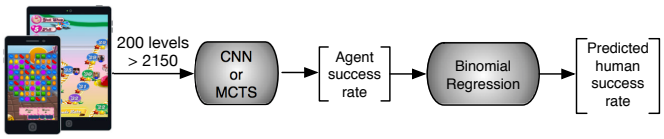
Binomial regression with random effects describes the data well but it is also possible to model the same data — log-transformed, with linear regression. The drawback with linear regression is that it lacks interpretation when the prediction goes over 1 or below 0. Also, because the relationship cannot be assumed to be strictly linear, as explained at the beginning of this point, the variance of the model is higher than the variance estimated with binomial regression.



(a) Train a CNN agent



(b) Fit a binomial regression model



(c) Predict human success rate for new levels

Fig. 3. Three flow charts describing the overview of our proposed approach.

IV. Experimental Evaluations

In this section, we evaluate the proposed approach on the games at hand. We compare MCTS results to the CNN results for *CCS* and additionally show prediction accuracy for *CCSS*. We did not build an MCTS agent for *CCSS* as this is a non-trivial and time-consuming task in a live game on a game engine not optimized for simulating agents. Thus, such a comparison to MCTS for *CCSS* is not possible. Therefore, we describe the details of the setup for the CNN agent in *CCS* and show results for *CCSS* where they apply. The overall approach for both games is identical.

A. Briefs of Human-Player Datasets

The data required to train the human-like agents is gathered via tracking the state-action pairs (samples) from approximately 1% of players, selected at random, during about 2 weeks. By the time when we collected the data, there were about 2,400 levels released in *CCS*. For training the CNN agents, we use 5,500 state-action pairs per level for the level range of 1 to 2,150, obtaining a dataset with nearly 1.2×10^7 samples. The data set is split into 3 subsets: training set (4,500 samples per level), validation set (500 samples per level), and test set (500 samples per level).

B. Evaluation Procedures and Settings

The experiment design is illustrated in Fig. 3. Prior to the recent few applications of deep learning in developing intelligent game agents (e.g. [11], [12], [32]), MCTS variants (e.g. [8], [25]–[27]) served as one of the mainstream approaches for simulating gameplay (as discussed in section II) and MCTS

TABLE I
THE SELECTION OF NETWORK ARCHITECTURES AND HYPER-PARAMETERS

Network Architecture	Searched Hyper-parameters	Validation Accuracy (%)
12Conv+ELU+GAP [a]	α, BS	**32.35**
12Conv+ReLU+FC+Dropout [b]	α, BS, p	25.72
12Conv+ReLU+GAP [c]	α, BS	28.59
20ResBlocks+ELU+GAP [d]	α, BS	30.01
Random Policy [*]	N/A	16.67

[a] The selected network architecture (Fig. 2a) that achieved the best validation accuracy (indicated in bold).
[b] A network consisting of 12 convolutional layers with ReLU activation functions, followed by 2 dropout regularized FC layers.
[c] Same as [a] except that ReLU is used in all convolutional layers.
[d] The convolutional layers used in [a] were replaced by 20 residual blocks of two convolutional operations and a shortcut connection around these two.
[*] Baseline: an entirely random policy for choosing game moves.

still plays a key role in many of the recent deep learning applications. Therefore we use Poromaa's implementation of MCTS [8] as a benchmark agent in our experiments. The implementation considers partial objective fulfillment when a roll-out does not lead to a win, instead of just binary values (win or loss). The MCTS agent is much more time consuming than the CNN agent. We want to understand how well the CNN agent does compared to the MCTS agent. The MCTS agent runs with 100 attempts on each level to get an estimate of $\rho_{\text{MCTS},i}$ but at the same time it runs 200 self-play simulations before taking a decision about which move to make for each position. In our comparison we run the CNN agent with 100 attempts and also with 1,000 attempts as a proxy for what could be more than 100,000 attempts if we want to compare the total number of simulations[2].

Training one version of our CNN model takes about 24 hours on a single machine with 6 CPUs and one Nvidia Tesla K80 GPU. Game-play simulations are executed using 32 CPUs in parallel. All computational resources are allocated on demand from a cloud service provider. The selected network architecture (Fig. 2a: 12Conv+ELU+GAP) is a result of the pre-study on data (state-action pairs) generated from game-play by MCTS agents. We experimentally evaluated 4 different network architectures, each of which requires a hyper-parameter search of learning rate (α), batch size (BS), and dropout keep probability (p). The values used when conducting a hyper-parameter grid search are respectively $\alpha \in \{5\times10^{-5},\ 1\times10^{-4},\ 5\times10^{-4}\}$, $BS \in \{2^7, 2^8, 2^9\}$, and $p \in \{0.4, 0.5, 0.6\}$. We report the best validation accuracy achieved by different network architectures in Table I. We found that a learning rate of 5×10^{-4} and a batch size of 2^9 lead to the best results.

C. The Training Performance of CNN-based Agents

From this section, we will base our analysis on the best performing architecture (i.e. 12Conv+ELU+GAP) in the previous section. Agents using that network architecture are trained with the data obtained by tracking human-players. The validation and test accuracy reached around 47% for CCS and 48% for CCSS. Comparing the validation accuracies we notice that CNN agents trained on real human-player data performed almost 50% better than the ones trained on data produced by MCTS agents. We tried to improve the accuracy by adding complexity to the model in form of more filters per layer as well as adding a layer of linear combination after GAP; but none of those added components made any significant improvement to the network's performance.

D. The Performance of Simulating Game Play

We can now use the trained CNN agent as a policy evaluating all actions $a \in \mathbf{A}$ given a state s, where $\mathbf{A}$ is the set of actions. The action with the highest probability $\max_{a\in\mathbf{A}} P(a|s)$ is then executed by the agent. The MCTS agents use 200 simulations to make one move in one state. This number proved in [8] to produce good results using a tolerable amount of time. Selecting and executing an action leads to a new state s' with a new set of possible actions $\mathbf{A}'$. The available actions are then again evaluated by the respective agents. This loop of executing an action given a state is continued until a terminal state is reached (either fulfillment of the objective or out of moves).

E. Comparing Predictions

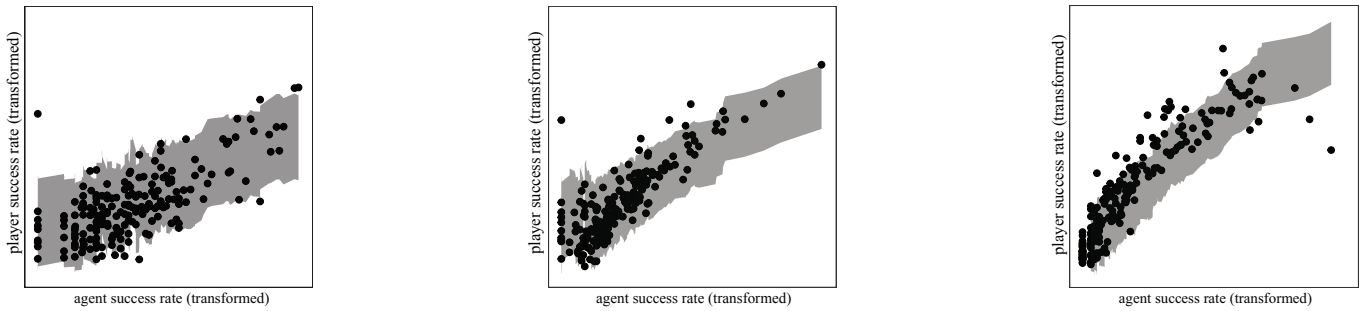
The predicted values for the 200 test levels and the associated prediction bands are shown in Fig. 4. The graphs compare prediction accuracy for the CNN agents and the MCTS agent. The CNN agents played both CCS and CCSS while the MCTS agent played only CCS. Additionally, it also shows the impact of the number of attempts on the prediction. For the CNN agents the prediction is based on 100 or 1,000 attempts and for the MCTS agent prediction is based on 100 attempts.

Table II summarizes the models. We see that MAE is lower for CCS than CCSS and that both CNN with 100 attempts and 1000 attempts has a lower MAE than MCTS. The prediction band is also much wider for MCTS indicating that the CNN agent is a stronger predictor. It is interesting to see that for the MCTS agent the ratio of prediction outside the 95% prediction band is close to the expected 5% and the out-of-band ratio is much higher for CNN. This is partly due to the wider prediction band for MCTS but it also suggests that the MCTS is quite robust to the evolution of the game. Note that the game is evolving with every new level, sometimes introducing new elements which the CNN has not trained on. Therefore, the CNN must be retrained when new elements are introduced to the game for optimal prediction performance but that was not done here. MCTS and any model predicting player difficulty measures would need to retrain their prediction model for new game elements but additionally, the CNN agents need new tracked data to update the policy.

V. FUTURE WORK

The policy that the CNN agent is learning is the average policy of all the players. It would improve difficulty predictions if we could learn different policies representing

[2]MCTS: 100 attempts, $\sim$ 30 moves per attempt, 200 simulations per move



(a) CCS: MCTS 100 Att/Lvl (b) CCS: CNN 1000 Att/Lvl (c) CCSS: CNN 1000 Att/Lvl

Fig. 4. The success rate obtained by different agents plotted against the success rate of human-players. Note, the values have been transformed. The scale is the same in all plots. The actual success rates for players are considered as sensitive information and therefore removed from the plot. The grey shaded areas indicates the 95% prediction band. The graphs provide visual overview of the model prediction performances. The uncertainty band is narrower for the CNN agents and thus the prediction for players' prediction performance is captured better.

TABLE II
OVERALL ESTIMATION PERFORMANCE OF 2 GAMES: *CCS* AND *CCSS*

Agent	Att/Lvl	Game	MAE	out-band ratio	STDDEV
MCTS	100	CCS	5.4%	4%	53%
CNN	1,000	CCS	4.0%	11%	35%
CNN	100	CCS	4.9%	24%	33%
CNN	1,000	CCSS	5.7%	17%	38%
CNN	100	CCSS	6.6%	23%	35%

different kind of players. Creating player "personas" based on different policies to represent clusters of similar players has the potential to greatly increase the understanding of levels. With different "personas" we could measure how often different policies agree and how certain the move predictions are. Possibly indicating the different experiences players have, e.g. if a level needs a high level of strategy or not. It could also improve the prediction to play non-deterministically with the CNN policy, with probabilities given by the CNN prediction output or ϵ-greedy. The architecture and hyper-parameters of the CNN can likely be improved which would be interesting to investigate further, especially with more data. Practically, it is important to measure other key metrics which can be done in a very similar way as the difficulty. We have already done this for score distribution and move distribution in CCS, CCSS and other games. For Procedural Content Generation (PCG) [47], the proposed agent can be a critical ingredient in the generation loop. For example, providing a fitness function for an evolutionary algorithm in a search-based approach [48]. Finally, we have indirectly been using the CNN agent for Quality Assurance. Playing with an agent which visits tens of thousands of the most relevant states in a level's state space has proven valuable and could prove to be an interesting research on its own.

VI. CONCLUSIONS AND PERSPECTIVES

Inspired by the recent advancement of deep learning techniques, mostly in the domain of computer vision, we proposed a framework for estimating level difficulty of match-3 games, the core of which is essentially a CNN-based agent trained on human-player data. However, the method is general and well suited for many games, in particular where content creation is sequential. The predictive power of our approach outperformed the state-of-the-art MCTS-based agents by a large margin on prediction accuracy and execution efficiency.

In CCS we can now estimate the difficulty of a new level in less than a minute and can easily scale the solution at a low cost. This compares to the previous 7 days needed with human playtesting on each new episode of 15 levels. This completely changes the level design process where level designers have now more freedom to iterate on the design and focus more on innovation and creativity than before. Internally, we have also tried this approach on a game in development using rather limited playtest data. Nevertheless, we were able to train a decent agent, albeit much noisier than in CCS and CCSS, which has helped a lot with the iterative process of game development. Since we ran the experiments presented in this paper we have used the CNN agent for more than a year, for more than 1,000 new levels in CCS. The prediction accuracy has been stable and when new game features have been presented it has been easy to retrain the agent to learn the new feature and continue predicting the difficulty.

REFERENCES

[1] J. Hamari, N. Hanner, and J. Koivisto, "Service quality explains why people use freemium services but not if they go premium: An empirical study in free-to-play games," *International Journal of Information Management*, vol. 37, no. 1, pp. 1449–1459, 2017.

[2] K. Alha, E. Koskinen, J. Paavilainen, and J. Hamari, "Free-to-Play Games: Professionals' Perspectives," in *Proceedings of Nordic Digra*. Gotland, Sweden, 2014, pp. 1–14.

[3] A. Denisova, C. Guckelsberger, and D. Zendle, "Challenge in digital games: Towards developing a measurement tool," in *Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, ser. CHI EA '17. New York, NY, USA: ACM, 2017, pp. 2511–2519. [Online]. Available: http://doi.acm.org/10.1145/3027063.3053209

[4] M. Seif El-Nasr, A. Drachen, and A. Canossa, *Game Analytics*. London: Springer, 2013.

[5] A. Drachen and A. Canossa, "Towards gameplay analysis via gameplay metrics," in *Proceedings of the 13th International MindTrek Conference: Everyday Life in the Ubiquitous Era on - MindTrek '09*. New York, USA: ACM Press, 2009, p. 202.

[6] A. Zook, E. Fruchter, and M. O. Riedl, "Automatic Playtesting for Game Parameter Tuning via Active Learning," *Foundations of Digital Games*, 2014.

[7] A. Isaksen, D. Gopstein, and A. Nealen, "Exploring game space using survival analysis," in *Proceedings of the 10th International Conference on the Foundations of Digital Games*. Pacific Grove, CA, 2015.

[8] E. R. Poromaa, "Crushing Candy Crush: Predicting Human Success Rate in a Mobile Game using Monte-Carlo Tree Search," Master's thesis, KTH Royal Institute of Technology, 2017.

[9] F. Silva, S. Lee, and N. Ng, "AI as Evaluator: Search Driven Playtesting in Game Design," in *Proceedings of AAAI*. Phoenix City, USA, 2016.

[10] C. Clark and A. Storkey, "Training deep convolutional neural networks to play go," in *International Conference on Machine Learning*. Lille, France, 2015, pp. 1766–1774.

[11] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 1 2016.

[12] K. Shao, D. Zhao, Z. Tang, and Y. Zhu, "Move prediction in Gomoku using deep learning," in *Proceedings of IEEE Youth Academic Annual Conference of Chinese Association of Automation (YAC)*. Hefei, China, 2017, pp. 292–297.

[13] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[14] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents." *J. Artif. Intell. Res.(JAIR)*, vol. 47, pp. 253–279, 2013.

[15] C. Holmgård, A. Liapis, J. Togelius, and G. N. Yannakakis, "Evolving personas for player decision modeling," in *2014 IEEE Conference on Computational Intelligence and Games*, pp. 1–8.

[16] ——, "Personas versus clones for player decision modeling," in *Entertainment Computing – ICEC 2014*, Y. Pisan, N. M. Sgouros, and T. Marsh, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 159–166.

[17] C. Holmgård, M. C. Green, A. Liapis, and J. Togelius, "Automated playtesting with procedural personas through MCTS with evolved heuristics," *CoRR*, vol. abs/1802.06881, 2018. [Online]. Available: http://arxiv.org/abs/1802.06881

[18] M. T. Omori and A. S. Felinto, "Analysis of motivational elements of social games: a puzzle match 3-games study case," *International Journal of Computer Games Technology*, vol. 2012, p. 9, 2012.

[19] C. Guckelsberger, C. Salge, J. Gow, and P. Cairns, "Predicting player experience without the player.: An exploratory study," in *Proceedings of the Annual Symposium on Computer-Human Interaction in Play*, ser. CHI PLAY '17. New York, NY, USA: ACM, 2017, pp. 305–315. [Online]. Available: http://doi.acm.org/10.1145/3116595.3116631

[20] N. Shaker, S. Asteriadis, G. N. Yannakakis, and K. Karpouzis, "Fusing visual and behavioral cues for modeling user experience in games," *IEEE Trans. Cybernetics*, vol. 43, no. 6, pp. 1519–1531, 2013. [Online]. Available: https://doi.org/10.1109/TCYB.2013.2271738

[21] D. Churchill, A. Saffidine, and M. Buro, "Fast heuristic search for rts game combat scenarios." in *Proceedings of the 8th AIIDE*. Palo Alto, California, 2012, pp. 112–117.

[22] D. Perez, S. Samothrakis, and S. Lucas, "Knowledge-based fast evolutionary MCTS for general video game playing," in *Proceedings of IEEE Conference on Computational Intelligence and Games (CIG)*. Dortmund, Germany, 2014, pp. 1–8.

[23] A. Zook, B. Harrison, and M. O. Riedl, "Monte-carlo tree search for simulation-based strategy analysis," in *Proceedings of the 10th Conference on the Foundations of Digital Games*, 2015.

[24] S. Devlin, A. Anspoka, N. Sephton, P. Cowling, and J. Rollason, "Combining gameplay data with monte carlo tree search to emulate human play," 2016. [Online]. Available: https://aaai.org/ocs/index.php/AIIDE/AIIDE16/paper/view/14003

[25] T. Imagawa and T. Kaneko, "Enhancements in monte carlo tree search algorithms for biased game trees," in *Proceedings of IEEE Conference on Computational Intelligence and Games (CIG)*. Tainan, Taiwan, 2015, pp. 43–50.

[26] A. Khalifa, A. Isaksen, J. Togelius, and A. Nealen, "Modifying MCTS for Human-like General Video Game Playing," in *Proceedings of IJCAI*. New York, USA, 2016, pp. 2514–2520.

[27] A. Mendes, J. Togelius, and A. Nealen, "Hyper-heuristic general video game playing," in *Proceedings of IEEE Conference on Computational Intelligence and Games (CIG)*. Santorini, Greece, 2016, pp. 1–8.

[28] T. S. Nielsen, G. A. Barros, J. Togelius, and M. J. Nelson, "General video game evaluation using relative algorithm performance profiles," in *Proceedings of European Conference on the Applications of Evolutionary Computation*. Copenhagen, Denmark, 2015, pp. 369–380.

[29] P. Bontrager, A. Khalifa, A. Mendes, and J. Togelius, "Matching games and algorithms for general video game playing," in *Proceedings of the Twelfth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE 2016, October 8-12, 2016, Burlingame, California, USA.*, 2016, pp. 122–128. [Online]. Available: http://aaai.org/ocs/index.php/AIIDE/AIIDE16/paper/view/14015

[30] R. Coulom, "Efficient selectivity and backup operators in Monte-Carlo tree search," *Computers and games*, vol. 4630, pp. 72–83, 2007.

[31] T. P. Runarsson and S. M. Lucas, "Preference learning for move prediction and evaluation function approximation in Othello," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 6, no. 3, pp. 300–313, 2014.

[32] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[33] P. Peng, Q. Yuan, Y. Wen, Y. Yang, Z. Tang, H. Long, and J. Wang, "Multiagent bidirectionally-coordinated nets for learning to play starcraft combat games," *arXiv preprint arXiv:1703.10069*, 2017.

[34] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of monte carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in games*, vol. 4, no. 1, pp. 1–43, 2012.

[35] M. Świechowski, H. Park, J. Mańdziuk, and K.-J. Kim, "Recent advances in general game playing," *The Scientific World Journal*, vol. 2015, pp. 1–22, 2015.

[36] S. Gelly and D. Silver, "Monte-carlo tree search and rapid action value estimation in computer go," *Artificial Intelligence*, vol. 175, no. 11, pp. 1856–1875, 2011.

[37] ——, "Combining online and offline knowledge in uct," in *Proceedings of the 24th International Conference on Machine learning*. Oregon, USA, 2007, pp. 273–280.

[38] A. Krizhevsky, I. Sutskever, and G. E. Hinton, *ImageNet Classification with Deep Convolutional Neural Networks*, 2012.

[39] P. Eisen, "Simulating human game play for level difficulty estimation with convolutional neural networks," Master's thesis, KTH Royal Institute of Technology, 2017.

[40] S. Purmonen, "Predicting game level difficulty using deep neural networks," Master's thesis, KTH Royal Institute of Technology, 2017.

[41] M. Lin, Q. Chen, and S. Yan, "Network in network," in *Proceedings of ICLR*. Banff, Canada, 2014.

[42] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *Information and Software Technology*, vol. 51, no. 4, pp. 769–784, 9 2014.

[43] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)," in *Proceedings of ICLR*. Vancouver, Cadana, 2016, pp. 1–13.

[44] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of International Conference on Machine Learning*. Lille, France, 2015, pp. 448–456.

[45] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*. Las Vegas, USA, 2016, pp. 770–778.

[46] J. Hinde and C. G. Demétrio, "Overdispersion: models and estimation," *Computational Statistics & Data Analysis*, vol. 27, no. 2, pp. 151–170, 1998.

[47] N. Shaker, J. Togelius, and M. J. Nelson, *Procedural Content Generation in Games*, 1st ed. Springer Publishing Company, Incorporated, 2016.

[48] V. Volz, J. Schrum, J. Liu, S. M. Lucas, A. Smith, and S. Risi, "Evolving Mario Levels in the Latent Space of a Deep Convolutional Generative Adversarial Network," *ArXiv e-prints*, May 2018.

A Critical Analysis of Punishment in Public Goods Games

Garrison W. Greenwood
Elect. & Computer Engr. Dept.
Portland State University
Portland, OR 97207–0751 USA
greenwd@pdx.edu

Hussein Abbass
School of Engr. and Info. Tech.
University of New South Wales
Canberra, ACT 2600, Australia
hussein.abbass@gmail.com

Eleni Petraki
Faculty of Arts and Design
University of Canberra
Canberra, ACT 2600, Australia
Eleni.Petraki@canberra.edu.au

***Abstract*—Social dilemmas arise whenever individuals must choose between their own self-interests or the welfare of a group. Economic games such as Public Goods Games (PGG) provide a framework for studying human behavior in social dilemmas. Cooperators put their self-interests aside for the group benefit while defectors free ride by putting their self-interests first. Punishment has been shown to be an effective mechanism for countering free riding in both model-based and human PGG experiments. But researchers always assume, since this punishment is costly to the punisher, it must be altruistic. In this study we show costly punishment in a PGG has nothing to do with altruism. Replicator dynamics are used to evolve strategies in a PGG. Our results show even a minority of punishers can improve cooperation levels in a population if the cooperators who punish are trustworthy. Finally, we argue punishment as a strategy in social dilemmas is never altruistic.**

***Index Terms*—altruism, costly punishment, public goods games, social dilemma**

I. Introduction

Cooperation is pervasive throughout nature but how it originates or why it persists is not fully understood. In the animal world cooperation is largely restricted to kin whereas in human populations non-kin cooperation is frequently observed in social dilemmas. Social dilemmas arise whenever individuals must decide whether to act in their own self-interest or in the best interest of a group. All social dilemmas have two conflicting properties [1]: (1) the group benefits more from mutual cooperation than mutual defection, and (2) individuals can always do better by defecting regardless of what others might do. Temptation to defect is strong so most social dilemmas end with everyone defecting.

Many social dilemmas involve the management of public goods. A public good is anything that requires contributions to create it and everyone benefits from its existence whether or not they contributed. Continued contributions are necessary to sustain the public good. Contributions can take many forms including time, effort or money. For example, taxes fund national defense and everyone benefits from it whether or not they pay taxes. Charities, such as the Salvation Army, depend on volunteers and donations of money or goods that can be resold; the charity then provides services to anyone in need whether or not they contributed. Other examples include public broadcasting, public land usage (parks, hiking trails, etc.) and pollution control. The fundamental question is what is the incentive to contribute anything if individuals can still benefit without contributing?

Economic games such as the *public goods game* (PGG), which is an N-player version of a prisoner's dilemma game, has been widely used to help get some answers. During each round of a PGG players simultaneously choose whether to cooperate by contributing money to a public fund while defectors contribute nothing. The fund is then increased and distributed equally among all players regardless of whether or not they contributed. Cooperators must subtract their contribution so defectors always do better. If cooperation and defection are the only strategies, and there is no opportunity to punish free riders, then the inevitable outcome is mutual defection. Of course if everyone defects then there are no contributions, the fund has no money and everyone loses (a Nash equilibrium). PGGs have been used in both computer simulations and human experiments to help gain insight into how cooperation in a population might evolve over time [2], [3], [4], [5] .

Several ideas have been posited on how cooperation evolves in populations [6], although some of them such as indirect reciprocity or kin selection don't provide proximate explanations when the interactions are repeated or if participants aren't related. Punishment has been shown to promote cooperation in PGGs although not always. The idea behind punishment is by imposing a penalty on free riders (defectors), such as a fine, will convince them that cooperation will be more beneficial. Usually this punishment is considered altruistic—i.e., it is costly to the punisher and no future material gain from imposing the punishment is expected [7].

In this paper we study altruistic punishment in a PGG in depth. Our results show that if the punishment is sufficiently high enough, then cooperators can successively saturate the population. In particular, we show punishment can be effective even if punishers are in the minority. However, this punishment does require punishers to be trustworthy which means if they initially commit to punish, they will continue to punish until all defectors are purged.

We also argue that, even though punishment can be an effective deterrent against defection, it may actually not be altruistic at all. Consequently, the way punishment is implemented in existing models and human experiments may be

978-1-5386-4359-4/18/$31.00 ©2018 IEEE

unrealistic which can make finding proximate explanations for cooperation more difficult.

The paper is organized as follows. Section II describes our model with simulation results presented in Section III. Section IV provides an explanation on the nature of effective punishment and makes the case why altruistic punishment may not exist in human populations.

II. The PGG Model

$N > 2$ players participate in a finite but unknown number of rounds of a PGG. Each round consists of two phases: a contribution phase and a punishment phase. In the contribution phase $m \leq N$ players choose to cooperate by contributing an amount c to a public fund while the remaining players defect and contribute nothing. An external benefactor then multiplies the fund by an amount r where $1 < r < N$ and this increased fund is equally distributed amongst the N players. Thus each player receives an amount rmc/N. Defectors are free riders who exploit cooperators because cooperators must subtract their contribution c; defectors always get a higher return.

In the punishment phase $n \leq m$ cooperators punish the defectors by imposing a penalty β on them. This penalty is subtracted from the defector's return and is intended to make free riding less profitable. Punishment is costly because there is a cost γ for imposing this punishment. The payoff π in each round to an individual x_i is thus

$$\pi(x_i) = \begin{cases} rmc/N - c & \text{cooperator who doesn't punish} \\ rmc/N - c - \gamma & \text{cooperator who does punish} \\ rmc/N - \beta & \text{defector} \end{cases} \tag{1}$$

Each cooperator imposes β/n of the penalty. The punishment cost γ is fixed regardless of the punishment levied. (Of course β = 0 if n=0.)

Discrete replicator equations are used to predict how strategies evolve in the population. Let k_i^j denote the number of players who choose strategy $i \in \{C, D\}$ at iteration j. Then the frequency of strategy i is $p_i^j = k_i/N$. At iteration $j+1$ the frequency becomes

$$p_i^{j+1} = p_i^j \left(\frac{f_i}{\bar{f}} \right) \tag{2}$$

where f_i is the fitness of strategy i and $\bar{f}$ is the average population fitness. If the quantity in parenthesis is greater than 1.0 then strategy i increases in the next iteration; decreases if the quantity is less than 1.0 and doesn't change if the quantity equals 1.0. Here we assume the payoffs given in Eq. (1) equates to fitness. The cooperator fitness value used is the payoff of cooperators who do not punish. (This issue will be discussed in detail in Section IV.) Plotting p_C^j shows how cooperation levels change over time.

Multiplying both sides of Eq. (2) by N yields the equivalent expression

$$k_i^{j+1} = k_i^j \left(\frac{f_i}{\bar{f}} \right) \tag{3}$$

The quantity in parenthesis on the right hand side must be be quantized to make sure only integer values for k_i^{j+1} are produced. The quantization method described below is one we have used previously [8]. The algorithm below returns k_i' which is the new integer value of k_i in the next iteration.

1) Compute
$$k_i' = \left\lfloor Np_i + \tfrac{1}{2} \right\rfloor \quad , \quad N' = \sum_i k_i'$$
2) Let $d = N' - N$. If $d = 0$, then go to step 4. Otherwise, compute the errors $\delta_i = k_i' - Np_i$.
3) If $d > 0$, decrement the d k_i''s with the largest δ_i values. If $d < 0$, increment the $|d|$ k_i''s with the smallest δ_i values.
4) Return $[k_1' \quad k_2']$ and exit.

III. Results

All simulations are run with $N = 40$ players. During the first phase of each round cooperators (C) contribute an amount $c = 1$ to a common fund which is then multiplied by a factor $r = 3$ and split evenly among all N players. The initial population contains $m = 30$ (75%) C players. Figure 1 shows the results for various punishments under replicator dynamics. When $\beta = 0.0$ there is no punishment (blue plot) which makes the population quickly evolve to an all-D state. The C strategy becomes dominant whenever $\beta > 1.25$. Consequently, when $\beta = 1.3$ (red plot) the punishment is enough to get the population to evolve to an all-C state. At an intermediate value of $\beta = 0.9$ (yellow plot) the punishment is insufficient to make defectors switch strategies. This shows that punishment values not sufficiently high enough only delay the undesirable mutual defection population state.

The second experiment is designed to see if a minority of cooperators can reverse the trend towards mutual defection. All C players agree that they will not punish during the second phase of each round. However, should the frequency of cooperators drop below 50% then they will all punish the defectors during the second phase. The amount of punishment inflicted by each of the m remaining cooperators is $\beta = (0.1{+}1.25)/m$ which gives a penalty slightly above the 1.25 minimum. Figure 2 show that with a sufficiently high enough penalty it is possible to recover and achieve an all-C population.

But this raises an interesting question. The reversal depicted in Figure 2 presumes all C players are trustworthy. That is, if they initially agree to punish, then they continue to punish until all D players are purged. Punishment is costly so any C player who punishes pays a cost γ which is subtracted from his return. Players may be tempted to stop punishing if it looks like the population is evolving towards an all-C state. Untrustworthy players still cooperate; they just stop punishing to avoid paying the additional cost γ.

To test the effect of untrustworthy players we allowed n players to stop punishing once the cooperator frequency reached 80%. Figure 3 indicates a small number of untrustworthy players can be tolerated (about 10%). However, at higher levels the cooperation frequency starts dropping and the rate

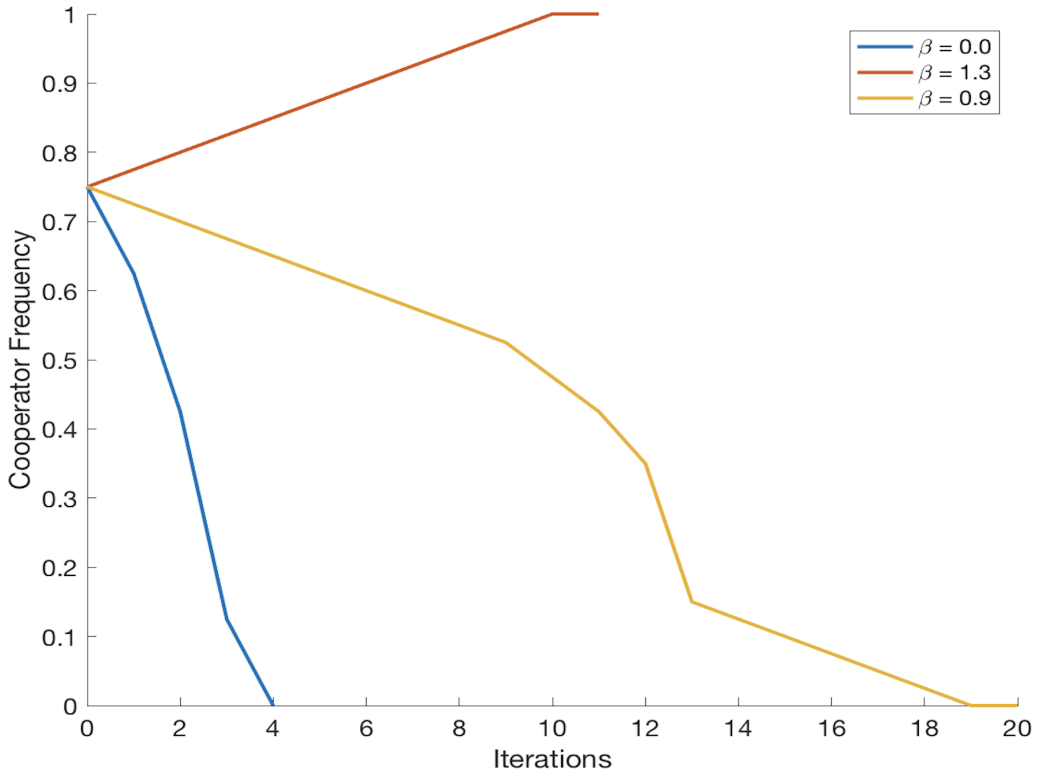


Fig. 1.

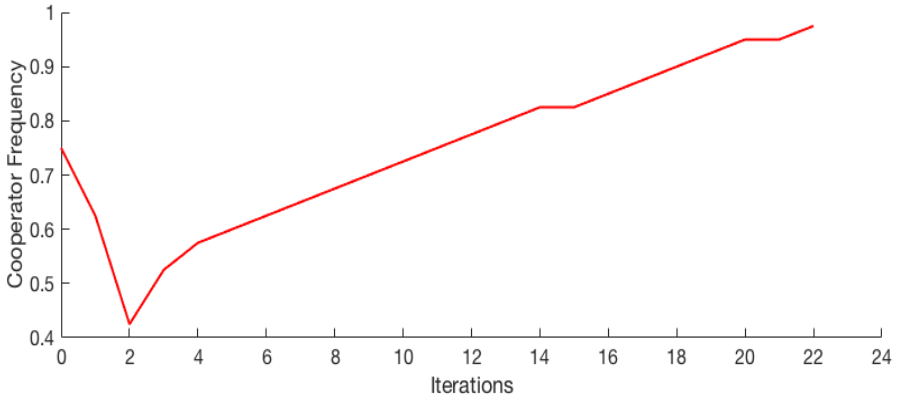


Fig. 2.

of decrease depends on how many untrustworthy players are present.

IV. Discussion

The simulation results show punishment can be an effective deterrent to defection, even when cooperators are in the minority, so long as that punishment is high enough. Insufficient punishment makes no sense because it merely delays the inevitable all-D case. (See Figure 1.) By Eq. (1) cooperator net returns will be higher than defector net returns if $\beta > c$. It is not necessary for the penalty to be more than $c + \gamma$ (the total costs paid by a cooperator who punishes). The objective is to make cooperation more attractive than defection—not cooperation with punishment more attractive. Moreover, cooperators who punish agreed to do so prior to playing the first round. Defectors did not participate in those discussions and therefore should not be expected (or required) to become punishers.

In model-based investigations the normal practice is to set r, c, β and γ and then run the simulation to see how the cooperator frequencies changes [9], [10], [11]. Choosing β a priori—at least in real-world social dilemmas—is unrealistic. As shown in Figure 1 picking punishment levels too low

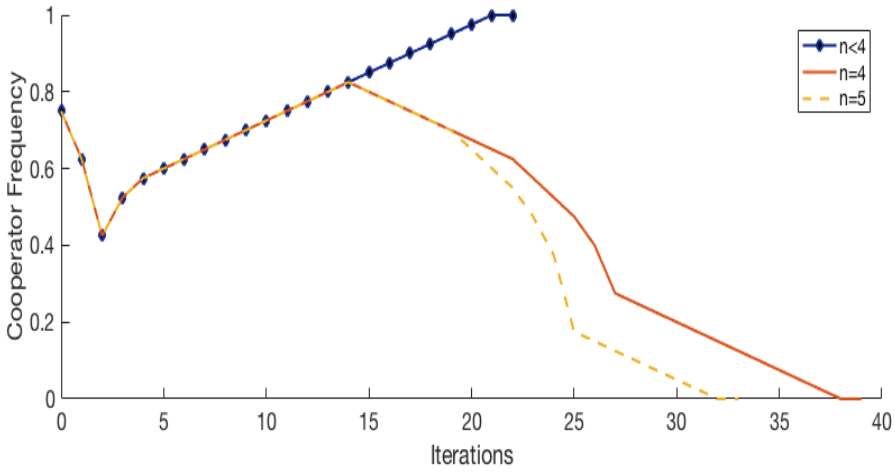


Fig. 3.

accomplishes nothing except delaying the inevitable all-D state. Cooperators thus have two choices: either punish to try and stop further defection or accept the inevitable outcome of mutual defection. For example, consider commercial fisherman operating in the Tasman Sea. Cooperators limit their catch to keep the fish population viable while defectors maximize their profit by taking as many fish as possible. Cooperator fisherman can either impose punishment to stop the over-fishing or resign themselves to seeing the fish population go extinct. Punishment levels therefore must be set based on the number of available punishers currently in the population or it will not be effective.

Conventional wisdom is the origin of altruistic punishment is complicated and warrants further investigation. Why would some individuals be willing to bear a cost for punishing when others might be the beneficiaries of lower defection levels? Fehr and Gächter [12] believe such punishment constitutes a second order public good. The decision to punish is actually quite simple in real-world situations: either punish to stop defection or accept the consequences of mutual defection. Human experiments designed to investigate altruistic punishment typically have players compete for only 6–10 rounds [12]. This means previously conducted human experiments probably have not been formulated appropriately to study altruistic punishment because they were not run long enough to evolve towards an all-D state. It would be interesting to see if players would be more inclined to punish as the all-D state is approached.

Fehr and Fischbacher [7] listed two components of altruistic punishment: (1) punishers may not necessarily receive any future material gain, and (2) the punishment is costly. There are problems with both of these being attributed to altruistic punishment, at least in the context of economic social dilemmas. Any belief that the first component is necessary for punishment to be altruistic is at best naive. Punishers intentionally punish; they expect their punishment will convince defectors to stop defecting. All participants know a priori the more cooperators in the population, the higher the return for both cooperators who punish and those that don't. Consequently, punishers do anticipate future higher returns because they expect the punishment will be effective.

In our simulations the punishment cost γ played no role in determining how the cooperation frequency changes. The rationale behind this approach comes from the motivation to punish. Human experiments suggest punishers punish defectors because they have little tolerance for what is perceived to be unfair behavior. For example, Fehr and Gächter [12] found negative emotions—specifically anger—being a prime motivator to punish. Seip et. al [13] put it more forcefully "...anger provides the fuel by which people are willing to spend costly personal resources to ensure that defectors get their due". If negative emotions, rather than altruism, are the reason why individuals decide to punish then any associated costs become irrelevant and may even be ignored.

Conventional wisdom believes punishment in economic social dilemmas is altruistic. In our view that belief is incorrect. Punishment does exist in human interactions but it is not altruistic. Researchers in both human experiments and model-based studies claim the punishment is altruistic *because* it is costly. In other words, punishers are willing to pay a personal cost for the benefit of the group. But cost doesn't really matter whenever anger is the motivation behind a free rider's punishment. Moreover, this cost may very well have have nothing to do with the group interests but more with self-interest—which is the complete opposite of altruism. Every defector who switches to a cooperator benefits the group including the punisher. Long-term interests can (and often do) override short-term interests. The punisher more likely is willing to bear the cost of punishment because the alternative results in disastrous personal financial outcomes for himself.

In the end it doesn't matter if the desire to punish is motivated by anger against free riders or economic survival, the punishment cost becomes irrelevant. Punishment is not altruistic if the punisher doesn't care about the cost or if his

own financial prosperity takes precedence over the welfare of the group.

V. Future Work

Our results show that effective punishment depends on punishers being trustworthy. That is, players who are willing to punish remain true to their word and do not decide to stop punishing. There is a temptation to stop punishing if it appears punishment is increasing cooperation levels. The idea of having a dilemma within a dilemma is intriguing. This concept is known as a "nested social dilemma". To our knowledge nested social dilemmas have not been studied in N-player environments. There is an opportunity to look at the PGG and the N-Player trust game we did studied previously [14] and to combine them together. Specifically, the question is whether altruistic punishment could be a measure of one type of trust. This will the focus of our future work.

References

[1] R. Dawes. Social dilemmas. *Ann. Rev. Psychol.*, 31:169–193, 1980.

[2] C. Hauert, A. Traulsen, H. Brandt, M. Nowak, and K. Sigmund. Via freedom to coercion: the emergence of costly punishment. *Science*, 316(5833):1905–1907, 2007.

[3] K. Jaffe. Altruism, altruistic punishment and social investment. *Acta Biotheor.*, 52(3):155–172, 2004.

[4] N. Nikiforakis and H. Normann. A comparative statics analysis of punishment in public-good experiments. *Exper. Econ.*, 11(4):358–369, 2008.

[5] B. Rockenbach and M. Milinski. The efficient interaction of indirect reciprocity and costly punishment. *Nature*, 444(7120):718–723, 2006.

[6] M. Nowak. Five rules for the evolution of cooperation. *Science*, 314(5805):1560–1563, 2006.

[7] E. Fehr and U. Fischbacher. The nature of human altruism. *Nature*, 425(6960):785–791, 2003.

[8] G. Greenwood, H. Abbass, and E. Petraki. Finite population trust game replicators. In *Proc. 2016 Australasian Conf. on Artifical Life and Comput. Intell.*, LNAI 9592, pages 324–335, 2016.

[9] X. chen, A. Szoinoki, and M. Perc. Probabilistic sharing solves the problem of costly punishment. *New Jour. of Phys.*, 16:083016, 2014.

[10] J. Fowler. Altruistic punishment and the origin of cooperation. *Proc. Nat'l. Acad. Sci.*, 102(19):7047–7049, 2005.

[11] A. Hintze and C. Adami. Darwinian evolution of cooperation via punishment in the "public goods" game. In *Proc. Alife XII Conf.*, pages 445–450, 2010.

[12] E. Fehr and S. Bächter. Altruistic punishment in humans. *Nature*, 415(10):137–140, 2002.

[13] E. Seip, W. Van Dijk, and M. Rotteveel. Anger motivates costly punishment of unfair behavior. *Motiv. Emot.*, 38:578–588, 2014.

[14] H. Abbass, G. Greenwood, and E. Petraki. The n-player trust game and its replicator dynamics. *IEEE Trans. on Evol. Comput.*, 20(3):470–474, 2016.

Monte-Carlo Tree Search for Implementation of Dynamic Difficulty Adjustment Fighting Game AIs Having Believable Behaviors

Makoto Ishihara
Graduate School of Information Science and Engineering
Ritsumeikan University
Shiga, Japan
is0153nx@ed.ritsumei.ac.jp

Suguru Ito
Graduate School of Information Science and Engineering
Ritsumeikan University
Shiga, Japan
is0202iv@ed.ritsumei.ac.jp

Ryota Ishii
Graduate School of Information Science and Engineering
Ritsumeikan University
Shiga, Japan
is0245ve@ed.ritsumei.ac.jp

Tomohiro Harada
College of Information Science and Engineering
Ritsumeikan University
Shiga, Japan
harada@ci.ritsumei.ac.jp

Ruck Thawonmas
College of Information Science and Engineering
Ritsumeikan University
Shiga, Japan
ruck@is.ritsumei.ac.jp

***Abstract*—In this paper, we propose a Monte-Carlo Tree Search (MCTS) fighting game AI capable of dynamic difficulty adjustment while maintaining believable behaviors. This work targets beginner-level and intermediate-level players. In order to improve players' skill while at the same time entertaining them, AIs are needed that can evenly fight against their opponent beginner and intermediate players, and such AIs are called dynamic difficulty adjustment (DDA) AIs. In addition, in order not to impair the players' playing motivation due to the AI's unnatural actions such as intentionally taking damage with no resistance, DDA methods considering restraint of its unnatural actions are needed. In this paper, for an MCTS-based AI previously proposed by the authors' group, we introduce a new evaluation term on action believability, to the AIs evaluation function, that focuses on the amount of damage to the opponent. In addition, we introduce a parameter that dynamically changes its value according to the current game situation in order to balance this new term with the existing one, focusing on adjusting the AI's skill equal to that of the player, in the evaluation function. Our results from the conducted experiment using FightingICE, a fighting game platform used in a game AI competition at CIG since 2014, show that the proposed DDA-AI can dynamically adjust its strength to its opponent human players, especially intermediate players, while restraining its unnatural actions throughout the game.**

***Index Terms*—Monte-Carlo tree search, dynamic difficulty adjustment, fighting game AI, believable, FightingICE**

I. Introduction

Fighting games are real-time games in which a character controlled by a human player or a game AI has to defeat their opponent using various attacks and evasion. In this work, AI is defined as a computer program that controls a character in a game. There are two types of matches in fighting games: Player VS Player (PvP), where two human players fight against each other, and Player VS Computer (PvC), where a human player fights against an AI-controlled character. An AI in PvC usually acts as the opponent for the human player who plays alone, sometimes as a sparring partner. In this work, we focus on PvC and target beginner and intermediate human players in fighting games.

One of the main features of beginner and intermediate players is that they do not fully know the game information such as character operations, available actions and fighting styles or tactics. They are often defeated by players who fully know about the game and by AIs that are too strong compared with them. This may cause beginner and intermediate players to lose the motivation to play the game, in the middle of improvement of their skill, and quit it. To prevent this, AIs are needed that can entertain beginner and intermediate players, while such players are still improving their playing skills.

Previously, the authors' group proposed a Monte-Carlo Tree Search (MCTS) fighting game AI called "Entertaining AI" (eAI) [1] whose goal is to entertain human players. This AI can evenly fight against its opponent players by dynamically adjusting its strength according to their playing skill, called dynamic difficulty adjustment (DDA). Namely, eAI will conduct an action according to the current game situation: when eAI is losing, it will conduct a strong action, otherwise, eAI will conduct a weak action. From the experimental results, eAI could entertain its opponent human players by evenly fighting against them. However, we observed that eAI often conducted unnatural actions such as repeating no-hit attacks and repeating step back even though the distance between the characters is far away. In order not to impair players' playing motivation due to AIs' unnatural actions such as those by eAI mentioned above, DDA methods able to restrain its unnatural actions are

978-1-5386-4359-4/18/$31.00 ©2018 IEEE

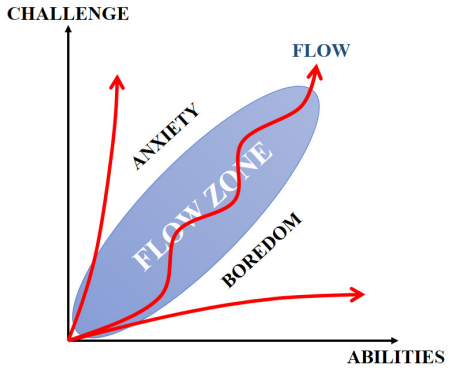


Fig. 1. Game flow [4]

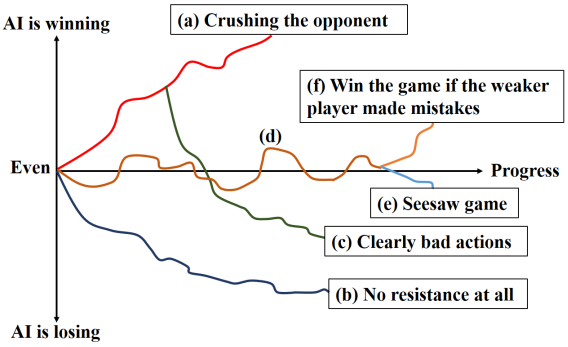


Fig. 2. Game flow in fighting games

needed [2].

In this paper, we propose an MCTS fighting game AI capable of dynamic difficulty adjustment while maintaining believable behaviors. This work targets beginner-level and intermediate-level players. We use eAI as a based AI and we introduce a new evaluation term on action believability, to the AI's evaluation function, that focuses on the amount of damage to the opponent. In addition, we introduce a parameter that dynamically changes its value according to the current game situation in order to balance this new term with the existing term in the evaluation function. We verify the performance of our proposed DDA-AI by a subjective experiment using FightingICE, a fighting game platform used in a game AI competition at CIG since 2014 [3].

II. Game Flow

Chen [4] mentioned the required elements by which players can enjoy playing games and how to design games to satisfy players using game flow (Fig. 1). In Fig. 1, the x-axis represents players' skill of the game and the y-axis represents the game difficulty. This figure indicates that players can enjoy playing the game if their skill and the game difficulty fall in "FLOW ZONE". That is, adjusting the game difficulty according to players' skill is needed. This can be said not only for game design, but also game AIs.

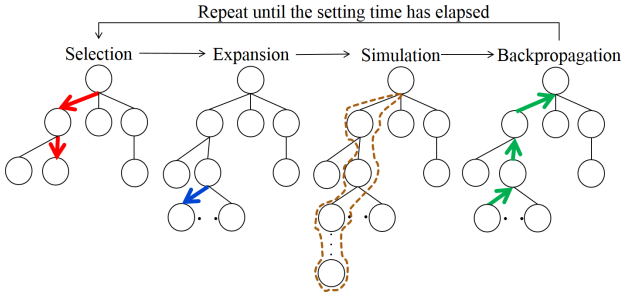


Fig. 3. An overview of MCTS

Ikeda and Viennot [2] mentioned the required elements according to which players can enjoy playing games and how to design them in terms of AIs in Go. They said using the aforementioned game flow that AIs are needed that can adjust their strength according to the opponent players' skill to evenly fight against or lose with a little difference in winning ratio. Fig. 2 shows the game flow applied to fighting game AIs by us with reference to the aforementioned work by Ikeda and Viennot. In Fig. 2, players cannot enjoy playing the game if the opponent AI crushes them (a) or loses with no resistance at all (b). Additionally, performing clearly unnatural actions only to balance the game (c) also impairs players' enjoyment. AIs should evenly fight against its opponent without unnatural actions (d), and finally, AIs might lose to its opponent with a little difference (e), or win if the opponent made some mistakes (f). That is, DDA-AIs capable of restraining its unnatural actions are needed.

III. Existing Methods for MCTS-Based DDA

In this section, we describe two DDA-AIs using MCTS. These AIs are used for comparison with our proposed AI.

A. Entertaining AI

Entertaining AI (eAI) was an MCTS-based DDA-AI proposed by our group [1]. This DDA method combines MCTS, Roulette Selection, and Rule-Based. In this section, we mainly explain MCTS, but we point out here that Roulette Selection, where the frequency of each action actually played by the opponent human player is used in simulation of his/her actions, is deployed in all of the AIs evaluated in this work. For more details about the other methods, please see Ishihara *et al.* [5].

Fig. 3 shows an overview of MCTS applied to fighting games. This MCTS is based on an open loop approach [6]. In this figure, the root node has the current game information which consists of both characters' Hit-Points (HPs), energies, positions, ongoing actions and the remaining time of the game. Each node except the root node represents an action. In this MCTS, an action spans from its input to its end, at which the next action becomes executable. An edge simply represents the connection between a parent node and its child node. When a parent node's action has finished, the next action will be one of its child nodes. In summary, the game tree using this MCTS represents the execution order of the AI's actions.

eAI repeats the four steps in Fig. 3 within a time budget of T_{max}. After the time budget is depleted, eAI selects the most visited direct child node (action) from the root node as the next action. The rest of this subsection explains each step of MCTS.

1) Selection: The child nodes with the highest Upper Confidence Bounds ($UCB1$) value [7] are selected from the root node until a leaf node is reached. The formula of $UCB1$ is:

$$UCB1_i = \overline{X}_i + C\sqrt{\frac{2\ln N}{N_i}}, \tag{1}$$

where N_i is the number of times node (action) i was visited, N is the sum of N_i for node i and its sibling nodes and C is a constant. $\overline{X_i}$ is the average evaluation of node i represented by the following formula:

$$\overline{X}_i = \frac{1}{N_i}\sum_{j=1}^{N_i} eval_j, \tag{2}$$

where $eval_j$ is the reward value gained in the jth simulation and is defined as:

$$eval_j = 1 - \tanh\frac{\left|afterHP_j^{my} - afterHP_j^{opp}\right|}{Scale}, \tag{3}$$

where $afterHP_j^{my}$ and $afterHP_j^{opp}$ stand for HP of the AI and the opponent after the jth simulation, respectively, and $Scale$ is a constant. As the HP difference between the AI and the opponent after the simulation is closer to 0, $eval_j$ will obtain an evaluation value closer to 1. Thereby, strong actions are highly evaluated when the AI is losing; otherwise, weak actions.

eAI selects the nodes with the highest $UCB1'_i$ value (using $\overline{X}_i$ value normalized by using formula (4)) from the root node until a leaf node is reached.

$$\overline{X}'_i = \frac{\overline{X}_i - \overline{X}_{min}}{\overline{X}_{max} - \overline{X}_{min}} \tag{4}$$

In this formula, $\overline{X}_{max}$, $\overline{X}_{min}$ stand for the maximum and minimum $\overline{X_i}$ in all nodes at the same tree depth.

2) Expansion: After a leaf node is reached in the $Selection$ part, if the number of times the leaf node is explored exceeds a threshold N_{max} and the depth of the tree is lower than a threshold D_{max}, all possible child nodes are created at once from the leaf node.

3) Simulation: A simulation is carried out for T_{sim} seconds, sequentially using all actions in the path from the root node to the current leaf node for the AI, and actions selected by Roulette Selection (see [5]) for the opponent. If the number of actions of the AI or the opponent used in the simulation is less than a given number, five in our previous work, randomly selected actions will be used after all actions of the AI or the opponent have been conducted. The variable $eval_j$ is then calculated using formula (2).

4) Backpropagation: $eval_j$ obtained from the simulation part is backpropagated from the leaf node to the root node. The $UCB1$ value of each node along the path is updated as well.

B. True Proactive Outcome-Sensitive Action Selection

True Proactive Outcome-Sensitive Action Selection (TPOSAS) is one of the MCTS-based DDA-AIs with believability proposed by Demediuk *et al.* [8]. TPOSAS also uses the same $UCB1$ formula (1). However, TPOSAS evaluates nodes using the following formula:

$$node.score = -\left(|h_s| - I_h\right)^+, \tag{5}$$

where h_s is the HP difference between the AI and the opponent, I_h defines the interval within which all HP differences can be neglected, and $(\cdot)^+$ indicates the ramp function, i.e., a function behaving like the identity function for positive numbers and returning 0 for negative numbers.

In this formula, the evaluations of all actions having h_s less than I_h will be 0; otherwise, will be negative. Therefore, all nodes (actions) with h_s less than I_h are more visited. Because there exist multiple actions that have the highest evaluation value of zero, unnatural behaviors like repeating the same action can be avoided.

In our experiment, I_h is set to 10% of the maximum player health as in the work by Demediuk *et al.* [8].

C. Problems

As we mentioned in Section I, eAI could entertain its opponent human players by evenly fighting against them. However, we could observe that eAI often conducted unnatural actions such as repeating no-hit attacks and repeating step back even though the distance between both characters is far away, especially in the game situation where the HP difference is around zero. In that situation, the evaluations of actions which do not give damage to the opponent and at the same time receive no damage such as moving actions will be higher than other actions. From this, one can readily see that eAI tends to select such unnatural actions in the above situation.

Demediuk *et al.* conducted the experiments where TPOSAS fought against human players and other AIs that were submitted to the Fighting Game AI Competition (FTGAIC)[1] to verify the method's effectiveness. From these experimental results, TPOSAS could dynamically adjust its strength according to its opponents' skill. However, although they mentioned about its believability, the authors did not quantitatively evaluate this factor. Also, they only used the HP difference at the end of the game as the evaluation criterion of DDA, and did not evaluate whether the AI can dynamically adjust its strength throughout the game.

IV. Proposed Method

In this section, we define what is believability in fighting games and explain our new DDA method considering fighting-game believable behaviors.

[1] http://www.ice.ci.ritsumei.ac.jp/~ftgaic/index.htm

A. Definition of Believable Behaviors in Fighting Games

As mentioned in Section I, the main purpose of fighting games is to defeat the opponent using various attacks and evasion. For that purpose, in this work, believable behaviors are defined as the aggressive behaviors aimed to defeat the opponent such as hitting attacks to the opponent properly. Conversely, unnatural behaviors are defined as those behaviors contrary to the main purpose mentioned above such as no-hit attacks (described in Section III-C), although it could be argued that such non-aggressive actions are also performed to a certain extent by some human players to taunt their opponents.

B. Evaluation Function with Believability

The new evaluation function taking into account believability is defined as follows:

$$eval_j = (1 - \alpha)\, B_j + \alpha E_j, \tag{6}$$

where E_j is for difficulty adjustment defined using the same formula as formula (3). B_j about the AI's aggressiveness (believability) represented by the following formula:

$$B_j = \tanh \frac{beforeHP_j^{opp} - afterHP_j^{opp}}{Scale}, \tag{7}$$

where $beforeHP_j^{opp}$ and $afterHP_j^{opp}$ stand for HP of the opponent before and after the jth simulation, respectively, and $Scale$ is a constant. If the AI gives a high amount of damage to the opponent, B_j will obtain a high evaluation value. Therefore, this term makes the evaluations of aggressive actions aimed at defeating the opponent higher than non-aggressive ones.

The coefficient α in formula (6) is dynamically determined by formula (8) based on the current game situation:

$$\alpha = \frac{\tanh\left(\frac{beforeHP_j^{my} - beforeHP_j^{opp}}{Scale}\right) + 1}{2}, \tag{8}$$

where $beforeHP_j^{my}$ and $beforeHP_j^{opp}$ stand for HP of the AI and the opponent, respectively, before the jth simulation, and $Scale$ is a constant. The more the AI is winning against the opponent, the closer α reaches 1. Conversely, the more the AI is losing against the opponent, α becomes closer to 0. Therefore, this coefficient makes it easier for the AI to select actions suitable for difficulty adjustment (E_j) when the AI is winning and select those increasing its aggressiveness (B_j) when the AI is losing. Also, when the HP difference is zero which means the AI is evenly fighting against the opponent, α becomes 0.5. In that situation, the AI selects actions that maintain both difficulty and believability.

In summary, the mechanism of our proposed method is making the AI select actions by considering not only how to adjust its difficulty toward the opponent's skill but also always how to defeat it.

V. Experiments

In this section, we describe the conducted experiments to verify the performance of our proposed DDA-AI (Believable Entertaining AI: BEAI).

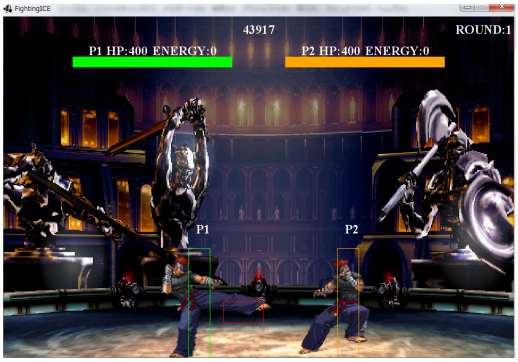


Fig. 4. Screen shot of FightingICE

TABLE I
Parameters used in the experiments

Notation	Meaning	Value
C	Balancing parameter	0.42
N_{max}	Threshold of the number of visits	7
D_{max}	Threshold of the tree depth	3
T_{sim}	The number of simulations	60 frames
T_{max}	Execution time of MCTS	16.5 ms
$Scale$	Scaling parameter	30

A. FightingICE

FightingICE (Fig. 4) is a real-time 2D fighting game platform used in a game AI competition (FTGAIC) at CIG since 2014 [3]. This game has all main elements of fighting games. In addition, it does not use a ROM emulator and has been originally developed from scratch and publicly made available for research purpose (see [10-14] for other recent publications using this platform), so there are no legal issues to be concerned. In FightingICE, one game consists of three 60-second rounds and one frame is set to 1/60 seconds. Each AI has to decide and input an action in one frame. Each character's initial HP is set to HP_{max}, and it will decrease when the character is hit. After 60 seconds or either character's HP is 0, the game will proceed to the next round, and each character's HP will be reset to HP_{max}. The character with the larger remaining HP at the end of the round is the winner.

In our experiments, the value of HP_{max} is set at 400 according to the rule of Standard Track of FTGAIC.

B. Parameters

The parameters used in our experiments are shown in Table I. These parameters were set empirically through pre-experiments.

C. Methods

We conducted subjective experiments to verify whether BEAI can adjust its strength according to the opponents' skill while maintaining its believability. We used 38 subjects (average age: 23.4 ± 2.2) in our experiments. Before starting our experiments, we conducted an informed consent session about our experiments, and subjects' consents were obtained with their signature in a separate informed consent form. In

addition, we used eAI and TPOSAS for comparison. Our experiments were conducted for two days; the first day is to measure each subject's skill of fighting games (Exp. 1) while the second day is to have them individually fight against eAI, TPOSAS and BEAI (Exp. 2). The content of Exp. 1 and Exp. 2 is given below.

1) Measurement of fighting games' skill (Exp. 1):
The procedure of Exp. 1 is as follows:

1) Explain the experiments and how to operate the character in FightingICE.
2) Ask each participant to fight against a non-action-AI for five minutes as practice.
3) Ask each participant to fight against an MCTS-Based high-performance AI (MctsAi) for one game.
4) Ask each participant to answer a questionnaire.
5) Repeat Steps 3 and 4 two times.

At Step 3, we used a sample AI of FTGAIC proposed by Yoshida *et al.* [9]. The questionnaire used at Step 4 is shown in Table II. This questionnaire was made with reference to previous studies [15] and [16]. We asked the subjects to evaluate each question in a 5-Likert scale (1: Strongly Disagree, 2: Disagree, 3: Neither, 4: Agree, 5: Strongly Agree). The evaluation of each factor is the average of the evaluation values of all questions (two in our case) belonging to each factor.

After finishing Exp. 1, we divided all subjects into three groups (G1, G2 and G3). We then confirmed that there is no significant difference between three groups in terms of both the average HP difference against the MctsAi and the evaluation value of the Challenge factor, using a Kruskal-Wallis test.

2) Fighting against eAI, TPOSAS and BEAI (Exp. 2):
The procedure of Exp. 2 is as follows:

1) Explain the experiments and how to operate the character in FightingICE.
2) Ask each participant to fight against a non-action-AI for five minutes as practice.
3) Ask each participant to fight against an AI for one game.
4) Ask each participant to answer a questionnaire.
5) Repeat Step 3 and 4 for all AIs

The questionnaire used in this experiment is the same as the one in Exp. 1, rather than rank-based questionnaires [17] where participants are asked to compare the play session they have just finished with the former one. This is because in our case the time window between the two consecutive play sessions is up to 3 minutes – one game – by which participants might not be able to make precise comparison. The fighting order of each AI was determined according to the Latin-square method as follows:

- G1 eAI→TPOSAS→BEAI
- G2 TPOSAS→BEAI→eAI
- G3 BEAI→eAI→TPOSAS

In Exp. 2, we evaluated each AI's performance using a metric called Average HP Difference Throughout the Game (AHDTG), described in Section VI-D, and the evaluation values of three factors in the questionnaire. Video clips showing typical gameplay by participants against these AIs are available[1].

TABLE II
CONTENT OF QUESTIONNAIRE

Dimension	Index	Content
Positive Affect	1	I felt it content
	2	I felt it enjoyable
Challenge	3	I felt it challenged
	4	I felt it stimulated
Believability	5	The opponent's attack skills were believable
	6	The opponent's dodging skills were believable

TABLE III
SUBJECT GROUPING IN TERMS OF FIGHTING GAME SKILL

Name	Average	Median	# of people
Expert	3 ± 38	4	11
Intermediate	-151 ± 29	-159	12
Beginner	-225 ± 25	-221	15

D. Average HP Difference Throughout the Game

AHDTG is introduced by the authors to evaluate how the AI can dynamically adjust its difficulty according to the opponent throughout the game, defined by the following formula:

$$AHDTG = \frac{\sum_{i=1}^{F_{total}} |HP_i^{my} - HP_i^{opp}|}{F_{total}}, \tag{9}$$

where HP_i^{my} and HP_i^{opp} stand for HP of the AI and the opponent at the frame i, respectively, and F_{total} stands for the total number of frames in this round. If the AI evenly fight against the opponent throughout the round, the value of AHDTG becomes small. This indicates that the smaller the value of AHDTG is, the more the AI can dynamically adjust its difficulty according to the opponent's skill throughout the round.

VI. RESULTS AND DISCUSSIONS

In this section, we show the experimental results and our discussions in terms of AHDTG and each factor of the questionnaire. Note that the symbols $*$ and $**$ used in figures and tables in this section represent a significant difference at 5% and 1%, respectively.

A. Subject grouping

From the result of Exp. 1, we divided subjects into three groups –Expert, Intermediate and Beginner– based on the

[1]http://www.ice.ci.ritsumei.ac.jp/˜ruck/dda-cig2018.htm

TABLE IV
RESULTS OF A FRIEDMAN TEST ON AHDTG IN EACH GROUP

Name	p-value
Expert	.078
Intermediate	$.017^{*}$
Beginner	$.006^{**}$

TABLE V
RESULTS OF A FRIEDMAN TEST ON POSITIVE AFFECT IN EACH GROUP

Name	p-value
Expert	.658
Intermediate	.084
Beginner	.723

average HP difference at the end of the game against the MctsAi, using the k-means method with $k = 3$. Table III shows the result of subject grouping. In Table III, the column Average represents the aforementioned average HP difference and the standard deviation of subjects belonging to each group, each playing three games.

B. AHDTG

Fig. 5 shows the average AHDTGs against eAI, TPOSAS and BEAI in each group. In Fig. 5, the x-axis represents the group names, the y-axis represents the value of AHDTG, and the error bars represent standard deviations of AHDTG for the three AIs in each group. We can see that BEAI obtains less AHDTG than eAI and TPOSAS against Intermediate and Beginner. According to our analysis of the gameplay, BEAI tends to behave aggressively especially in the game situation where the HP difference is around zero, due to the new evaluation term about its aggressiveness, compared with other two AIs. Therefore, one can consider that compared to eAI which tends to behave strangely like intentionally filling up the HP difference after the value becomes too large (AI losing too much), BEAI could evenly fight against the opponent like a seesaw game shown in Fig. 2 (d).

However, BEAI obtains more AHDTG than eAI and TPOSAS against Expert. According to our observation, some players in Expert adopted a fighting style like "counter-attack" by which they appropriately hit their attacks to the opponent against the opponent's conducted actions. We could often see that these players hit their strong attacks such as the ultimate attack to BEAI when it stepped forward in order to shorten the distance; in other words, they exploited BEAI's aggressive behaviors against the AI. For this reason, BEAI couldn't adjust its strength against expert players compared with the other two AIs.

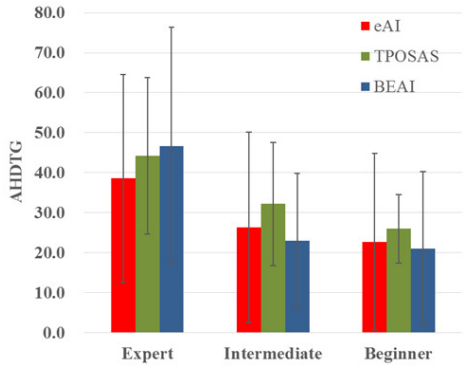


Fig. 5. Average AHDTGs against eAI, TPOSAS and BEAI, in each group

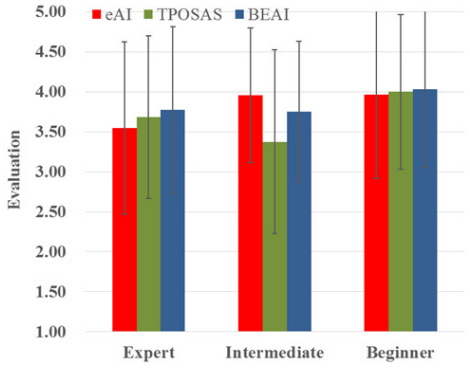


Fig. 6. Average evaluations of Positive Affect toward gameplay against eAI, TPOSAS and BEAI, in each group

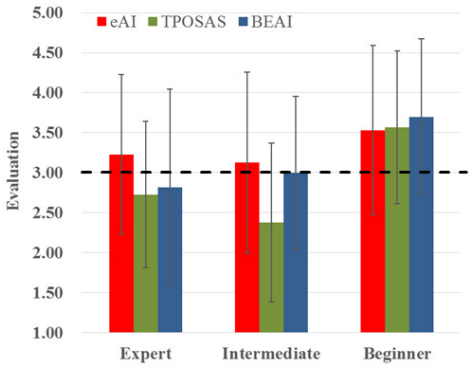


Fig. 7. Average evaluations of Challenge toward gameplay against eAI, TPOSAS and BEAI, in each group

Table IV shows the results of a Friedman test on AHDTG in each group. There are significant differences at 5% and 1% between the three AIs in Intermediate and Beginner, respectively. From these results, we can conclude that BEAI could dynamically adjust its difficulty against intermediate and beginner players throughout the game compared to the existing DDA methods.

C. Positive Affect

Fig. 6 shows the average evaluations of Positive Affect toward gameplay against eAI, TPOSAS and BEAI, in each group. In Fig. 6, the x-axis represents the group names, the y-axis represents the evaluation value (1: Boring $\sim$ 5: Enjoyable) of Positive Affect, and the error bar represents the standard deviation of it in each group. We can see that BEAI obtains higher evaluation values than eAI and TPOSAS against Expert and Beginner. However, it obtains a lower evaluation value than eAI against Intermediate. From our analysis, we could observe that BEAI often forced players to fight in the close range compared to the other two AIs. Subjects belonging to Intermediate fought against their opponent AIs using various actions and strategies, similar to those players in

TABLE VI
RESULTS OF A FRIEDMAN TEST ON CHALLENGE IN EACH GROUP

Name	p-value
Expert	.187
Intermediate	.024*
Beginner	.840

TABLE VII
RESULTS OF A FRIEDMAN TEST ON BELIEVABILITY IN EACH GROUP

Name	p-value
Expert	.886
Intermediate	.042*
Beginner	.420

Expert. However, since their skill is not that high, compared to Expert, they couldn't fight the way they wanted because of the BEAI's aggressive behavior compared to eAI, which led to the decrease in affect evaluation toward gameplay against BEAI. Although having this issue, BEAI obtains more than 3.75 points in all groups. Thus, we can still say that the subjects evaluated fighting against BEAI favorably.

Table V shows the results of a Friedman test on Positive Affect in each group. There is no significant difference between the three AIs in all groups. From these results, although there is no significant difference, we can conclude that BEAI could entertain expert and beginner players more than the existing DDA methods.

D. Challenge

Fig. 7 shows the average evaluations of Challenge toward gameplay against eAI, TPOSAS and BEAI, in each group. In Fig. 7, the x-axis represents the group names, the y-axis represents the evaluation value (1: Too weak $\sim$ 3: Good difficulty $\sim$ 5: Too strong; note that 3 is the best) of Challenge, and the error bar represents standard deviation for each AI in each group. We can see that BEAI obtains higher evaluation values than eAI and TPOSAS against Expert and Intermediate. However, BEAI obtains lower evaluation than the other two AIs against Beginner. From our analysis, we could observe that subjects belonging to Beginner often used simple strategies such as stepping forward and punching or kicking. As BEAI behaves aggressively, there were many situations where subjects and BEAI gave damage to each other in close ranges. Thus, they evaluated BEAI to be too strong due to these situations, compared to the other two AIs.

Table VI shows the results of a Friedman test on Challenge in each group. There is a significant difference at 5% between the three AIs in Intermediate. From these results, we can conclude that BEAI could adjust its difficulty against expert and intermediate players in a way that they felt the opponent AI's difficulty was suitable for them.

E. Believability

Fig. 8 shows the average evaluations of Believability toward gameplay against eAI, TPOSAS and BEAI, in each group. In Fig. 8, the x-axis represents the group names, the y-axis

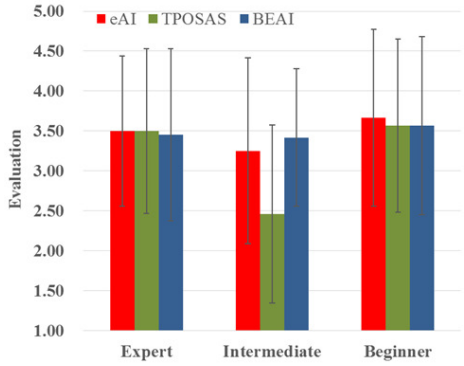


Fig. 8. Average evaluations of Believability toward gameplay against eAI, TPOSAS and BEAI, in each group

represents the evaluation value (1: Unnatural $\sim$ 5: Believable) of Believability, and the error bar represents the standard deviation for each AI in each group. We can see that BEAI obtains higher evaluation than eAI and TPOSAS against Intermediate. From our analysis, we could observe that BEAI conducted less unnatural actions as mentioned in Section III-C, especially the game situations where the HP difference is around zero. Thus, our proposed evaluation function could dynamically adjust the AI's difficulty while restraining its unnatural actions, and improve the evaluation value of Believability evaluated by intermediate players.

Table VII shows the results of a Friedman test on Believability in each group. There is a significant difference at 5% between the three AIs in Intermediate. From these results, we can conclude that BEAI could adjust its difficulty while restraining its unnatural actions against intermediate players.

VII. CONCLUSIONS AND FUTURE WORK

In order to improve players' skill while at the same time entertaining them, AIs are needed that can evenly fight against their opponent beginner and intermediate players; such AIs are called DDA-AIs. In addition, in order not to impair the players' playing motivation due to the AI's unnatural actions, DDA methods that can restrain their unnatural actions are needed. In this paper, we proposed an MCTS fighting game AI capable of DDA while maintaining its believable behaviors, targeting beginner-level and intermediate-level players. We used eAI proposed previously by our group [1] as a based AI (eAI) and introduced a new evaluation term on action believability, to the AI's evaluation function, that focuses on increasing the amount of damage to the opponent. In addition, we introduced a parameter that dynamically changes its value according to the current game situation in order to balance this new term with the existing term in the evaluation function.

From our experimental results, our proposed DDA-AI showed the best performance in terms of average HP difference throughout the game (AHDTG), Challenge and Believability against intermediate players, and AHDTG against beginner players. As a result, we conclude that our proposed DDA-AI could dynamically adjust its strength to its opponent human

players' skill, especially intermediate players, while restraining its unnatural actions throughout the game. The proposed evaluation function (6) has a potential to be applied to MCTS-based AIs in other games to maintain the aggressiveness while shrinking the performance gap with the opponent human player, in particular when the AI is winning.

However, although our proposed DDA-AI was evaluated favorably by intermediate and beginner players, it could not significantly improve the evaluation value of Positive Affect, compared to eAI. For future work, we plan to develop a new mechanism for entertaining players while keeping its believability. It might also be interesting to combine the proposed DDA-AI with a mechanism that directly emulates human players [18]. In addition, we will also develop much stronger AIs as based AIs for new DDA-AIs that can adapt to expert players.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for their helpful comments. They would also like to thank their lab members, in particular, the FightingICE team members for their fruitful discussions. This research was partially supported by Strategic Research Foundation Grant-aided Project for Private Universities (S1511026), Japan.

REFERENCES

[1] M. Ishihara, T. Miyazaki, T. Harada, and R. Thawonmas, "Analysis of Effects of AIs and Interfaces to Players' Enjoyment in Fighting Games," *IPSJ Journal*, vol. 57, no. 11, pp. 2415-2425, 2016, (in Japanese).

[2] K. Ikeda and S. Viennot, "Production of Various Strategies and Position Control for Monte-Carlo Go - Entertaining human players," in *Proc. Computational Intelligence in Games (CIG)*, 8 pages, 2013.

[3] F. Lu, K. Yamamoto, L. H. Nomura, S. Mizuno, Y. Lee, and R. Thawonmas, "Fighting Game Artificial Intelligence Competition Platform," in *Proc. IEEE 2nd Global Conference on Consumer Electronics (GCCE)*, pp. 320-323, 2013.

[4] J. Chen, "Flow in games (and everything else)," *Communications of the ACM*, vol. 50, no. 4, pp. 31-34, 2007.

[5] M. Ishihara, T. Miyazaki, C. Y. Chu, T. Harada, and R. Thawonmas, "Applying and Improving Monte-Carlo Tree Search in a Fighting Game AI," in *Proc. 13th International Conference on Advances in Computer Entertainment Technology (ACE 2016)*, no. 27, 2016.

[6] D.P. Liebana, J. Dieskau, M. Hunermund, S. Mostaghim, S. Lucas, "Open Loop Search for General Video Game Playing," in *Proc. the 2015 Annual Conference on Genetic and Evolutionary Computation (GECCO' 15)*, pp. 337-344, 2015.

[7] L. Kocsis and C. Szepesvári, "Bandit Based Monte-Carlo Planning," in *Proc. European Conference on Machine Learning (ECML)*, pp. 282-293, 2006.

[8] S. Demediuk, M. Tamassia, W. L. Raffe, F. Zambetta, X. Li, and F. Mueller, "Monte Carlo Tree Search Based Algorithms for Dynamic Difficulty Adjustment," in *Proc. Computational Intelligence and Games (CIG)*, pp. 53-59, 2017.

[9] S. Yoshida, M. Ishihara, T. Miyazaki, Y. Nakagawa, T. Harada, and R. Thawonmas, "Application of Monte-Carlo Tree Search in a Fighting Game AI," in *Proc. IEEE 5th Global Conference on Consumer Electronics (GCCE)*, pp. 623-624, 2016.

[10] R. Ishii, S. Ito, M. Ishihara, T. Harada and R. Thawonmas, "Monte-Carlo Tree Search Implementation of Fighting Game AIs Having Personas," in *Proc. 2018 IEEE Conference on Computational Intelligence and Games (CIG 2018)*, 2018.

[11] X. Neufeld, S. Mostaghim, and D. Perez-Liebana, "HTN fighter: Planning in a highly-dynamic game," in *Proc. 2017 Computer Science and Electronic Engineering (CEEC 2017)*, Colchester, pp. 189-194, Sep. 2017.

[12] S. Yoon and K.-J. Kim, "Deep Q Networks for Visual Fighting Game AI," in *Proc. 2017 IEEE Conference on Computational Intelligence and Games (CIG 2017)*, 2017.

[13] M.-J. Kim and K.-J. Kim, "Opponent Modeling based on Action Table for MCTS-based Fighting Game AI," in *Proc. 2017 IEEE Conference on Computational Intelligence and Games (CIG 2017)*, 2017.

[14] D.T.T Nguyen, V. Quang and K. Ikeda, "Optimized Non-visual Information for Deep Neural Network in Fighting Game," in *Proc. 9th International Conference on Agents and Artificial Intelligence (ICAART 2017)*, pp. 676-680, 2017.

[15] W. A. IJsselsteijn, K. Poels, and Y. De Kort, "The Game Experience Questionnaire: Development of a Self-report Measure to Assess Player Experiences of Digital Games", *Technical University Eindhoven, FUGA Technical Report*, 46 pages, 2007.

[16] B. Soni and P. Hingston, "Bots trained to play like a human are more fun," in *Proc. International Joint Conference on Neural Networks*, pp. 363-369, 2008.

[17] G. N Yannakakis and J. Hallam, "Evolving opponents for interesting interactive computer games," in *Proc. the 8th International Conference on the Simulation of Adaptive Behavior (SAB04); From Animals to Animats 8*, pp. 499-508, 2004.

[18] S. Devlin, A. Anspoka, N. Sephton, P.I. Cowling,"Combining Gameplay Data with Monte Carlo Tree Search to Emulate Human Play," in *Proc. Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE 2016)*, pp. 16-22, 2016.

Monte-Carlo Tree Search Implementation of Fighting Game AIs Having Personas

Ryota Ishii
Graduate School of Information Science and Engineering
Ritsumeikan University
Shiga, Japan
is0245ve@ed.ritsumei.ac.jp

Suguru Ito
Graduate School of Information Science and Engineering
Ritsumeikan University
Shiga, Japan
is0202iv@ed.ritsumei.ac.jp

Makoto Ishihara
Graduate School of Information Science and Engineering
Ritsumeikan University
Shiga, Japan
is0153hx@ed.ritsumei.ac.jp

Tomohiro Harada
College of Information Science and Engineering
Ritsumeikan University
Shiga, Japan
harada@is.ritsumei.ac.jp

Ruck Thawonmas
College of Information Science and Engineering
Ritsumeikan University
Shiga, Japan
ruck@is.ritsumei.ac.jp

***Abstract*—In this paper, we propose a method for implementing a game AI with a persona using Monte-Carlo Tree Search (MCTS). Video games are now a powerful entertainment media not just for players but spectators as well. Since each spectator has personal preferences, customized spectator-specific gameplay is arguably a promising option to increase the entertainment value of video games streaming. In this paper, we focus on personas, which represent playstyles in the game, in particular fighting games. In order to create an AI player (character) with a given persona, we use a recently developed variant of MCTS called Puppet-Master MCTS, which controls all characters in the game, and introduce a new evaluation function, which makes each character take their actions according to the given persona, and roulette selection-based simulation to this MCTS. The results of a conducted experiment using FightingICE, a fighting game platform used in a game AI competition at CIG since 2014, show that the proposed method can make both characters successfully behave according to given personas, which were identified by participants – spectators – in the experiment.**

***Index Terms*—Monte-Carlo tree search, persona, play style, fighting game AI, FightingICE**

I. Introduction

In recent years, more than 100 million spectators watch gameplay videos (GPV) every month using streaming platforms such as Twitch and YouTube. This phenomenon suggests video games are a powerful entertainment media not just for players but spectators as well. Spectators can be divided into three groups. "Let's Play" (LP) is one of them where spectators enjoy watching GPVs streamed by live streamers [1]. Such spectators often look for GPVs that entertain themselves by judging GPVs according to not only how well the players play them but also how entertaining those GPVs are. As a result, it is desirable that game streamers must be able to generate customized spectator-specific GPVs that match their spectators' preferences. Due to a wide variety of spectators' preferences, it is, however, challenging for streamers to provide such GPVs. Recently, Thawonmas and Harada proposed a novel concept of called procedural play generation (PPG) [2]. Their goal is to generate GPVs automatically and recommend those GPVs to spectators according to their preferences. To realize this concept, one needs artificial intelligence (AI) methods for automatically generating GPVs with various contents and recommender systems for recommending GPVs to spectators according to their preferences. In this paper, we cope with the AI part.

In particular, we propose a method that can automatically generate various GPVs using Monte-Carlo Tree Search (MCTS) [3], [4]. We target fighting games, one of the game genres whose GPVs are often streamed by streamers and adopted as competitions in eSports. In addition, we focus on playstyles in fighting games; such styles are called personas [5]. In order to generate GPVs where each AI player (character) has a specific persona, we adopt a recently proposed MCTS called Puppet-Master MCTS (PM-MCTS) [6] that controls all characters in the game as a base method and introduce a new evaluation function that is based on the distance between both characters and their actions. We also introduce roulette selection to the simulation part in PM-MCTS to improve the simulation accuracy. We verify whether our proposed method can generate GPVs where the characters have personas by a subjective experiment using FightingICE, a fighting game platform used in a game AI competition at CIG since 2014 [7].

II. Related Work

A. Persona

According to Tychsen and Canossa [5], persona in the game context is the playstyle of a player. They analyzed playstyles

978-1-5386-4359-4/18/$31.00 ©2018 IEEE

of various game players, and those playstyles were used for testing game designs. Following this definition of persona, Holmgård *et al*. [8] proposed a method for implementing AIs that behave according to specified personas using MCTS. They classified five personas in "MiniDungeons 2", which is a turn-based rogue-like game. They also introduced an evaluation function for each persona to MCTS to make the AI behave according to the given persona. From their experimental results, they showed that their proposed AIs could play "MiniDungeons 2" according to the specified personas in terms of the AIs' action frequcies and differences in decision making. However, they did not quantitatively verify whether spectators could subjectively recognize and identify the AIs' personas.

B. Puppet-Master MCTS

PM-MCTS is a variant of MCTS, proposed by Ito *et al*. [6], that controls all characters in the game using only a single game tree. An open loop approach [9] is adopted in PM-MCTS. Figure 1 shows an overview of PM-MCTS applied to a two-player fighting game, such as FightingICE. In this tree, according to the open loop approach, each node represents an action choice for either of the character (circle: P1; square: P2) that can execute a new action. PM-MCTS builds such a tree starting from an initial state, defined by information such as the Hit-Point (HP), energy, coordinates, and action of each character and the game remaining time. Each edge represents the ongoing execution of two actions: one just started from the parent node and the other started earlier by the other character. Note that the branching factor at a given node in PM-MCTS is the number of actions of the respective character, not the combination of all possible actions from all characters. In addition, it is more straight-forward and efficient to build the opponent models for PM-MCTS than for multiple MCTs, one for each character.

PM-MCTS comprises five steps: selection, expansion, simulation, backpropagation, and decision making, the first four of which are the usual steps in MCTS algorithms. These five steps are described individually in the following subsections.

1) Selection: Nodes are selected from the root node until a leaf node is reached according to the given selection criterion. In our work, we use Upper Confidence Bounds ($UCB1$) value [10], which is widely used in this step of MCTS, defined by the following formula:

$$UCB1_i^{pl} = \overline{X}_i^{pl} + C\sqrt{\frac{2\ln N}{N_i}}, \tag{1}$$

where N_i is the number of times node (action) i was visited, N is the sum of N_i for node i and its sibling nodes, and C is a constant. $\overline{X}_i^{pl}$ defined in formula (2) is the average evaluation value of node i from the perspective of character pl, the one who can start executing an action at this node. It is worth reminding that every node of the tree contains the $UCB1$ values from the perspective of both characters, as well as a counter on how many times the node has been visited. When the AI selects a child node, it uses the $UCB1$ value of the character who can start the next action at its parent node.

$$\overline{X}_i^{pl} = \frac{1}{N_i}\sum_{j=1}^{N_i} Eval_j^{pl}, \tag{2}$$

where $Eval_j^{pl}$ is the reward value gained in the jth simulation from the perspective of character pl.

In this work, the AI selects the nodes with the highest $UCB1$ value from the root node until a leaf node is reached.

2) Expansion: After a leaf node is reached in the $Selection$ step, if the number of times the leaf node has been explored exceeds a threshold N_{max} and the depth of the tree is lower than a threshold D_{max}, all possible child nodes are created at once from the leaf node. If the root node is the only node in the tree, PM-MCTS creates all of its child nodes, ignoring above conditions. Each newly created child node represents the game state when either of the characters can start a new action after the character at the parent node has finished its action. Note that if both characters can start their action at a leaf node of interest, PM-MCTS creates child nodes for P1 first, and when this situation happens again it creates child nodes for P2, and this alternation is continued.

3) Simulation: A simulation is carried out for T_{sim} seconds, sequentially using all actions of both characters in the path from the root node to the current leaf node. If T_{sim} has not passed yet after those actions have been executed, a rollout will be carried out until T_{sim} runs out using randomly selected actions; in other words, T_{sim} limits the rollout depth in this case. Each character's $Eval_j$ is then calculated using formula (2).

4) Backpropagation: Each character's $Eval_j$ obtained in the $Simulation$ step is backpropagated from the leaf node to the root node. Each character's $UCB1$ value at each node along the path is updated as well.

5) Decision Making: PM-MCTS repeats the above four steps until the character who will be executing an action at the root node becomes available to do so ,i.e., the character's previous action has finished. The AI then chooses the character's action according to a given recommendation policy. In this work, it selects the edge (action) connected to the direct child node that has the highest $\overline{X}_i^{pl}$ from the root node as the next action. That child node will be used as the next root node and its sibling nodes will be pruned. This step is explicitly described here and shown in Fig. 1 to emphasize the reuse of tree structures, as often done in the open-loop approach.

III. Personas in Fighting Games

In this section, we give the definition of personas in fighting games. In this work, we define two fighting-game personas, i.e., *RushDown* and *Zoning*, and individually assign them a set of actions. Both action sets do not overlap and are fixed during gameplay. The details of them are described in the following subsections.

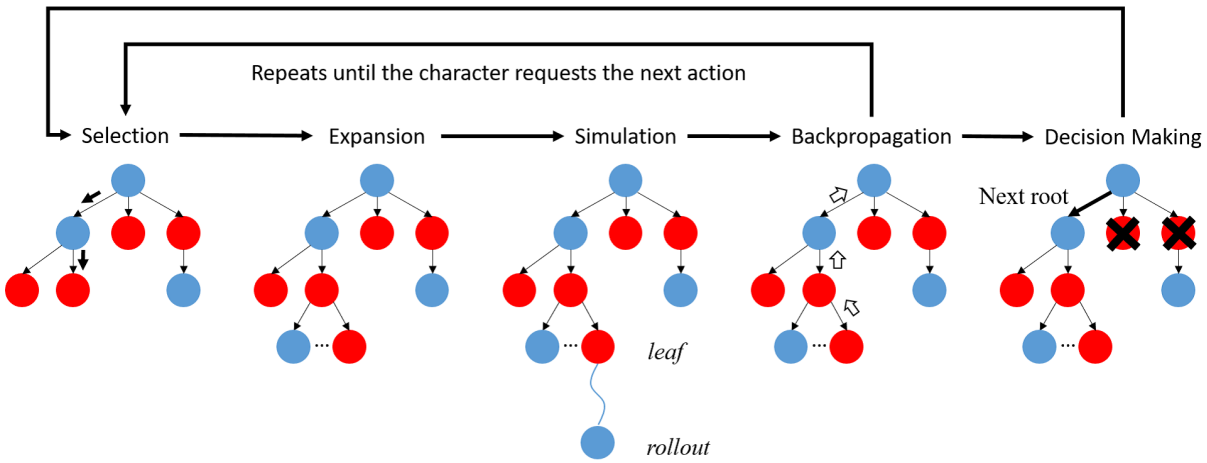


Fig. 1: An overview of PM-MCTS for a two-player game

A. *RushDown*

RushDown is a playstyle in fighting games where the player prefers to fight in close ranges. The behavioral tendency of *RushDown* is that the player often uses moving actions such as step forward to approach the opponent and close-range attacks such as punch or kick. In FightingICE, described in Section V, the actions shown in Table I are defined as actions belonging to *RushDown*.

B. *Zoning*

Zoning is a playstyle in fighting games where the player prefers to fight while keeping a certain distance with the opponent. The behavioral tendency of *Zoning* is that the player often uses moving actions such as back step to move away from the opponent, and long-range attacks such as fire-ball shoots. In FightingICE, the actions shown in Table II are defined as actions belonging to *Zoning*.

IV. Proposed Method

In this section, we describe the proposed method for automatically generating GPVs where each character behaves according to the given persona. PM-MCTS is used as a base method, but we introduce a new evaluation function and roulette selection-based simulation. The details of our proposed method are given in the following subsections.

A. *Evaluation function*

The proposed evaluation function for character pl in the jth simulation is defined as follows:

$$Eval_j^{pl} = ePersona_j^{pl} \times eHP_j^{pl}, \tag{3}$$

where $ePersona_j^{pl}$ is the evaluation value regarding how much the character's persona could be realized at a node of interest and is defined as

$$ePersona_j^{pl} = \frac{act_{Persona_j}^{pl} + dist_{Persona_j}^{pl}}{3} \tag{4}$$

TABLE I: All actions belonging to *RushDown*

Skill name	Skill content	Damage
STAND_A	Simple punch	5
STAND_B	Simple kick	10
CROUCH_A	Crouch punch	5
CROUCH_B	Crouch kick	10
STAND_D_DB_BA	Jumping punch	10
STAND_D_DB_BB	Sliding kick	20
FORWARD_WALK	Walk forward	0
DASH	Dash forward	0

TABLE II: All actions belonging to *Zoning*

Skill name	Skill content	Damage
STAND_D_DF_FA	Shoot projectile forward	10
STAND_D_DF_FB	Shoot strong projectile forward	40
THROW_A	Throw the opponent	10
THROW_B	Strongly throw the opponent	20
BACK_STEP	Step back	0
BACK_JUMP	Jump backward	0
FOR_JUMP	Jump forward	0

In the above formula, $act_{Persona_j}^{pl}$ is the term that evaluates whether character pl conducts an action belonging to the given persona, $Persona \in \{RushDown, Zoning\}$, in this simulation and is defined as

$$act_{Persona_j}^{pl} = \begin{cases} 1 & (belongs\ to\ Persona) \\ -1 & (otherwise) \end{cases} \tag{5}$$

If the character conducts an action belonging to $Persona$, it will obtain a positive evaluation; otherwise, a negative one.

The term $dist_{Persona_j}^{pl}$ considers the distance between both characters. In our work, according to our experience and preliminary experiments, we define $dist_{Persona_j}^{pl}$ when the persona of character pl is *RushDown* (formula (6)) and *Zoning* (formula (7)), respectively, as follows:

$$dist^{pl}_{RushDown_j} = \begin{cases} 2 & (distance < 130) \\ 2 - (\frac{distance}{width} \times 4) & (otherwise) \end{cases} \quad (6)$$

$$dist^{pl}_{zoning_j} = \begin{cases} -2 & (distance < 240) \\ 2 & (240 \leq distance < 450) \\ 2 - (\frac{distance}{width} \times 4) & (otherwise) \end{cases} \quad (7)$$

In formulas (6) and (7), $distance$ represents the distance between the characters, and $width$ is the number of pixels indicating the width of the game screen. When a character is within the distance range suitable for realization of the given persona, the highest evaluation value is obtained. In our work, we define the distance that a short-range attack hits the opponent as a suitable distance for ***RushDown*** while the distance that a long-range attack hits the opponent and that a character can avoid the opponent's close-range attacks as a suitable distance for *Zoning*.

In addition, in formula (3), eHP^{pl}_j is the term that evaluates how much the HP of the opponent of character pl has decreased and is defined as follows:

$$eHP^{pl}_j = oppHP_{root} - oppHP_{rollout} \quad (6)$$

where $oppHP_{root}$ and $oppHP_{rollout}$ stand for the HP of the opponent at the root node and after the rollout, respectively. If character pl gives a high amount of damage to the opponent, eHP will have a high value. The main role of this term is to keep the believability of character pl: if there was only $ePersona$ in our evaluation function, for example, a character might conduct unnatural actions such as repeating punch or dash only to realize *RushDown*, its persona.

The proposed evaluation function makes the character select actions by considering not only how to realize its persona but also always how to defeat the opponent, which is the main purpose of players in fighting games.

B. Roulette selection applied to rollout

The original PM-MCTS uses randomly selected actions for both characters in the rollout. However, if this was adopted in our work, actions not appropriate for each character's persona might be conducted. This may cause inaccurate evaluation of all nodes in the current path if the random-based rollout is still used.

In order to solve this issue, we introduce roulette selection, where the actions belonging to the persona of a character of interest have higher weights, to the rollout. The algorithm of roulette selection is shown in Algorithm 1. As the variables in this algorithm, $actionList$ is a list containing all actions, $actFit$ is an array containing all actions' fitness values, $dart$ is a threshold, and $totalFit$ is a sum of all actions' fitness values in $actFit$. The function initialize() initializes the fitness values of all actions passed to it as the argument $actionList$. In our work, the fitness value of each action is set to 3 if it belongs to the action set defined in the persona of a character of interest; otherwise, 1, not 0 in order to allow actions not belonging to this persona but with a strong damage value to be also chosen.

This mechanism makes a character of interest use actions that belong to its persona in a rollout more and hence increases the accuracy of the simulation.

Algorithm 1 Algorithm of roulette selection

```
// initializes all actions' fitness values
actFit ← initialize(actionList)
dart ← random(0,1)
wheel ← actFit[0]/totalFit
count ← 0
while dart > wheel do
    count ← count + 1
    wheel ← wheel + actFit[count]/totalFit
end while
//Returns the action selected by roulette selection
return ActionList.get(count)
```

Algorithm 2 Algorithm of the proposed method

```
state ← getNowState() //Gets the current game state
root ← Initialize(state) //Initializes the root node
while !isGameEnd do
    activePlayer ← root.activePlayer
    //Runs PM-MCTS until activePlayer can execute its
    next action
    bestAct ← TreeSearch(root, state, activePlayer)
    runAction(bestAct, activePlayer)
    root ← nextRoot(root, bestAct) //Changes the root node
    state ← getNowState() //Gets the current game state
end while
```

C. Algorithm of the proposed method

Algorithm 2 shows the algorithm of the proposed method. As the variables in this algorithm, $state$ represents the current game state, $root$ is the root node, $activePlayer$ is the character who will be executing the next action in the game, and $bestAct$ is the selected action by PM-MCTS. The function getNowState() is for obtaining the current game state, initialize() is for initializing a node passed to it as the argument, TreeSearch() is for running PM-MCTS, runAction() is for conducting an action passed to it as the argument, and nextRoot() is for changing the root node.

V. Experiment

In this section, we describe the conducted experiment to verify the performance of the proposed method (P-AI).

A. FightingICE

FightingICE (Fig. 2) is a real-time 2D fighting game platform used in a game AI competition (FTGAIC)[1] at CIG since

[1]http://www.ice.ci.ritsumei.ac.jp/˜ftgaic/
https://github.com/TeamFightingICE/FightingICE

Fig. 2: Screenshot of FightingICE

2014 [7] and for research [6, 11-23]. Because FightingICE has been originally developed from scratch without using a ROM emulator and publicly made available, there are no legal issues to be concerned. In FightingICE, one game consists of three 60-second rounds, and one frame lasts 1/60 seconds. Each character has to decide and input an action in one frame. The HP for both characters is initially set to HP_{max} and it decreases when the corresponding character is hit. When the play is conducted for 60 seconds or either of the two characters' HP becomes 0, the game will proceed to the next round unless the current round is the 3rd one, after which each character's HP will be reset to HP_{max}. The character with the larger remaining HP at the end of a round is the round's winner. In our experiment, the value of HP_{max} is set to 400 according to the rule of Standard Track of FTGAIC.

One of the limitations, from our work's perspective, of FightingICE used in FTGAIC is that each character must be individually controlled. To remove this limitation, we modified some functions in FightingICE so that PPM-MCTS can control both characters. Another limitation is that the characters can only obtain each time a game state delayed by 15 frames (0.25 s), taking into account the delay of human perception. However, since our work is focused on the generation of GPVs, not on the development of a character for fighting against another AI opponent or a human opponent in competitions, we also removed the delay from the FightingICE platform in our experiment.

B. AIs and parameters in use

In the experiment, we compared the proposed AI (P-AI) and another MCTS-based AI (M-AI) that controls only one character [18] using the same evaluation function and roulette selection described in Section IV. The parameters used in both AIs are shown in Table III. These parameters were set empirically through pre-experiments.

C. Details

Our experiment consists of 35 participants (average age: 22.8 ± 2.6).

TABLE III: Parameters used in the experiment

Notation	Meaning	Value
C	Balancing parameter	0.025
N_{max}	Threshold of the number of visits	10
D_{max}	Threshold of the tree depth	10
T_{sim}	Simulation-time budget	60 frames
$width$	Width of the game screen	960 pixels

TABLE IV: Persona of each character in a fight

P1	P2
RushDown	RushDown
Zoning	Zoning
RushDown	Zoning
Zoning	RushDown

1) Generation of GPVs: A 60-second GPV for each combination of two personas shown in Table IV was generated by each AI, leading to eight GPVs[2] in total. For generation of GPVs, the P-AI controlled both characters. However, since an M-AI can control only one character, two M-AIs (2M-AIs), each assigned a persona accordingly, were used to generate a GPV; at the simulation step, the opponent's actions are selected by roulette selection using its persona.

2) Procedure: The procedure of our experiment is as follows:

1) Explain the concept of each of the two personas, *RushDown* and *Zoning*, and show sample GPVs to a participant.
2) Ask each participant to watch one of the generated GPVs.
3) Ask each participant to choose the persona for P1 and that for P2, among "RushDown", "Zoning", and "Other".
4) Repeat Steps 2) and 3) for all generated GPVs.

Note that at Step 2), a GPV was displayed in random order.

VI. Results and Discussions

In this section, we show the experimental results and our discussions in terms of whether the participants were able to identify each character's persona in GPVs. The summarization tables of the correctly answering participants and incorrectly answering participants for both characters (a), P1 (b), and P2 (c) in each GPV are shown in Tables V–IX, in a 2×2 contingency table style often used in the McNemar's test conducted below. In these tables, P, M, T, F, and S are P-AI, 2M-AIs, the number of participants who correctly answered, the number of people who incorrectly answered, and the sum of numbers in the corresponding row or column, respectively. Note that in each table (a), we counted the number of participants who correctly answered the personas of both characters as T; otherwise, F. We describe the result of each persona combination in the following subsections.

A. RushDown vs RushDown

Table V shows the summarization of the numbers of correct participants and incorrect participants for the *RushDown* vs

[2]http://www.ice.ci.ritsumei.ac.jp/˜ruck/personaGPVs-cig2018.htm

RushDown GPVs generated by P-AI and 2M-AIs. In Tables Va and Vb, we can see that the participants could correctly answer the personas of both P1&P2 and P1 alone in the GPV generated by P-AI (33/35 or 94.3% in both cases) more than those in the one generated by 2M-AIs (P1&P2: 17/35 or 48.6%, P1: 18/35 or 51.4%). From the results of McNemar's tests, there are significant differences at 1% between P-AI and 2M-AIs in both cases. However, from Table Vc, both AIs obtain a high accuracy. From our observation, P2, controlled by either P-AI or 2M-AIs, aggressively shortened the distance between the characters even than P1 in the GPV generated by P-AI. Due to this behavior, P2's persona was judged as *RushDown* by most of the participants.

B. Zoning vs Zoning

Table VI shows the summarization of the numbers of correct participants and incorrect participants for the *Zoning* vs *Zoning* GPVs generated by P-AI and 2M-AIs. We can see that the participants could correctly answer the personas in the GPV generated by P-AI (P1&P2: 16/35 or 45.7%, P1: 31/35 or 88.6%, and P2: 20/35 or 57.1%) more than those in the one generated by 2M-AIs (P1&P2: 1/35 or 2.9%, P1: 4/35 or 11.4%, and P2: 12/35 or 34.3%). From the results of McNemar's tests, there are significant differences at 1% between P-AI and 2M-AIs for P1&P2 and P1, and at 10% for P2. However, there are fewer people who correctly answered both characters' personas than those who did not. This is due to the specification of attacks belonging to *Zoning*. All of the attacks in *Zoning* given in Table II need energy to conduct them, and this makes zoning characters select moving actions such as a jump forward. In addition, when both characters approach each other due to such moving actions, they often conduct actions that give damage to their opponent according to the term eHP in formula (6). Due to these behaviors, both characters' personas were judged as *RushDown* by a number of participants, especially in the GPV generated by 2M-AIs.

C. RushDown vs Zoning

Table VIII shows the numbers of correct participants and incorrect participants for the *RushDown* vs *Zoning* GPVs generated by P-AI and 2M-AIs. From these tables, we can see that the participants could correctly answer the personas in the GPV generated by P-AI more than those in the one generated by 2M-AIs for all cases, especially P1&P2 and P2. From the results of McNemar's tests, there are significant differences at 1% between P-AI and 2M-AIs for the three cases. This is because each M-AI individually decides actions according to its own evaluation function, without considering the opponent's next action, which is different from P-AI. This often caused mismatched fighting such as approaching each other, which looks like *RushDown* vs *RushDown*. Due to these behaviors, many participants answered wrong personas in the GPV generated by 2M-AIs.

D. Zoning vs RushDown

Table IX shows the summarization of the numbers of correct participants and incorrect participants for the *Zoning* vs *RushDown* GPVs generated by P-AI and 2M-AIs. In Tables Xa and Xb, we can see that the participants could correctly answer the personas of P1&P2 and P1 in the GPV generated by P-AI (P1&P2: 22/35 or 62.9%, P1: 31/35 or 88.6%) more than those in the one generated by 2M-AIs (P1&P2: 1/35 or 2.9%, P1: 2/35 or 5.7%). From the results of McNemar's tests, there is a significant difference at 1% between P-AI and 2M-AIs for each of the aforementioned two cases. However, from Table Xc, there are fewer correct participants for P2's persona in the GPV generated by P-AI than that by 2M-AIs, with a significant difference at 10%. From our observation, in the GPV generated by P-AI, P2 sometimes used projectile attacks belonging to *Zoning*. This is because such attacks, if used, can give more damage than simple close-range attacks belonging to *RushDown*, causing the term eHP for P-AI to obtain a much higher evaluation value than $ePersona$ and hence making rushdown characters behave like *Zoning* rather than *RushDown*.

E. Summary of the results

In summary, we can conclude that the participants could better identify both characters' personas in the GPVs generated by P-AI than those in the GPVs generated by 2M-AIs. As

TABLE V: The numbers of correct participants and incorrect participants for the *RushDown* vs *RushDown* GPVs generated by P-AI and 2M-AIs

(a) P1 & P2

P \ M	T	F	S
T	17	16	33
F	0	2	2
S	17	18	35

(b) P1

P \ M	T	F	S
T	18	15	33
F	0	2	2
S	18	17	35

(c) P2

P \ M	T	F	S
T	33	1	34
F	1	0	1
S	34	1	35

TABLE VI: The numbers of correct participants and incorrect participants for the *Zoning* vs *Zoning* GPVs generated by P-AI and 2M-AIs

(a) P1 & P2

P \ M	T	F	S
T	1	15	16
F	0	19	19
S	1	34	35

(b) P1

P \ M	T	F	S
T	4	27	31
F	0	4	4
S	4	31	35

(c) P2

P \ M	T	F	S
T	7	13	20
F	5	10	15
S	12	23	35

TABLE VIII: The numbers of correct participants and incorrect participants for the *RushDown* vs *Zoning* GPVs generated by P-AI and 2M-AIs

(a) P1 & P2

P \ M	T	F	S
T	4	28	32
F	1	2	3
S	5	30	35

(b) P1

P \ M	T	F	S
T	20	12	32
F	1	2	3
S	21	14	35

(c) P2

P \ M	T	F	S
T	8	25	33
F	1	1	2
S	9	26	35

TABLE IX: The numbers of correct participants and incorrect participants for the *Zoning* vs *RushDown* GPVs generated by P-AI and 2M-AIs

(a) P1 & P2

P \ M	T	F	S
T	1	21	22
F	0	13	13
S	1	34	35

(b) P1

P \ M	T	F	S
T	2	29	31
F	0	4	4
S	2	33	35

(c) P2

P \ M	T	F	S
T	21	3	24
F	10	1	11
S	31	4	35

mentioned earlier, P-AI controls both characters based on the information on their future actions. Due to this mechanism, P-AI can select the next action for each character that well expresses its persona.

VII. Conclusions and Future Work

Video games are now an attractive entertainment media not just for players but also spectators. To provide customized spectator-specific GPVs that match various kinds of spectators' preferences, AIs that can automatically generate GPVs with a variety of contents are needed. In this paper, we focused on personas, which represent playstyles, in fighting games. In order to generate GVPs where each character plays according to a given persona, we adopted a recently developed variant of MCTS called Puppet-Master MCTS, which controls all characters in the game, and introduced a new evaluation function and roulette selection-based simulation to this MCTS.

The results of the conducted experiment confirmed that the proposed AI can make both characters successfully behave according to their personas, which were identified by the participants or spectators in the experiment. However, the proposed AI still has an issue when controlling characters having the persona of *Zoning*. For future work, we plan to develop new evaluation functions and mechanisms to realize given personas more accurately. In addition, we plan to introduce more personas besides *RushDown* and *Zoning* for generating a higher variety of GPVs. It might also be interesting to consider believability [23] and human-play emulation [24] aspects or to analyze generated gameplay with action metrics recently proposed by Zook and Riedl [25]. We also plan to verify whether GPVs generated by our proposed AI can entertain spectators through extensive user studies.

Acknowledgment

The authors would like to thank the anonymous reviewers for their helpful comments. They would also like to thank their lab members, in particular, the FightingICE team members for their fruitful discussions. This research was partially supported by Strategic Research Foundation Grant-aided Project for Private Universities (S1511026), Japan.

References

[1] T. Smith, M. Obrist and P. Wright, "Changes the (Video) Game," in *Proc. 11th European Conference on Interactive TV and Video (Live-Streaming)*, ACM, pp. 131-138, 2013.

[2] R. Thawonmas and T. Harada, "AI for Game Spectators: Rise of PPG," in *Proc. AAAI 2017 Workshop on What's next for AI in games*, San Francisco, USA, pp. 1032-1033, 2017.

[3] R. Coulom, "Efficient selectivity and backup operators in monte-carlo tree search," in *Proc. International Conference on Computers and Games*, pp. 72-83, 2006.

[4] C.B. Browne, E. Powley, D. Whitehouse, S.M. Lucas, P.I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A Survey of Monte Carlo Tree Search Methods," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, pp. 1-43, 2012.

[5] A. Tychsen and A. Canossa, "Defining personas in games using metrics," in *Proc. 2008 Conference on Future Play: Research, Play, Share*, ACM, pp. 73-80, 2008.

[6] S. Ito, M. Ishihara, M. Tamassia, T. Harada, R. Thawonmas and F. Zambetta, "Procedural Play Generation According to Play Arcs Using Monte-Carlo Tree Search," in *Proc. 18th International Conference on Intelligent Games and Simulation*, pp. 67-71, 2017.

[7] F. Lu, K. Yamamoto, L. H. Nomura, S. Mizuno, Y. Lee and R. Thawonmas, "Fighting Game Artificial Intelligence Competition Platform," in *Proc. IEEE 2nd Global Conference on Consumer Electronics (GCCE)*, pp. 320-323, 2013.

[8] C. Holmgård, A. Liapis, J. Togelius, and G. Yannakakis, "Monte-Carlo Tree Search for Persona Based Player Modeling," in *Proc. AIIDE workshop on Player Modeling*, 7 pages, 2015.

[9] D.P. Liebana, J. Dieskau, M. Hunermund, S. Mostaghim, S. Lucas, "Open Loop Search for General Video Game Playing," in *Proc. the 2015 Annual Conference on Genetic and Evolutionary Computation (GECCO' 15)*, pp. 337-344, 2015.

[10] P. Auer, N. Cesa-Bianchi, P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine Learning*, vol. 47, no. 2-3, pp. 235-256, 2002.

[11] X. Neufeld, S. Mostaghim, and D. Perez-Liebana, "HTN fighter: Planning in a highly-dynamic game," in *Proc. 2017 Computer Science and Electronic Engineering (CEEC 2017)*, Colchester, pp. 189-194, Sep. 2017.

[12] S. Demediuk, M. Tamassia, Wi. Raffe, F. Zambetta, X. Li and F.F. Mueller, "Monte Carlo Tree Search Based Algorithms for Dynamic Difficulty Adjustment," in *Proc. 2017 IEEE Conference on Computational Intelligence and Games (CIG 2017)*, 2017.

[13] S. Yoon and K.-J. Kim, "Deep Q Networks for Visual Fighting Game AI," in *Proc. 2017 IEEE Conference on Computational Intelligence and Games (CIG 2017)*, 2017.

[14] M.-J. Kim and K.-J. Kim, "Opponent Modeling based on Action Table for MCTS-based Fighting Game AI," in *Proc. 2017 IEEE Conference on Computational Intelligence and Games (CIG 2017)*, 2017.

[15] D.T.T Nguyen, V. Quang and K. Ikeda, “Optimized Non-visual Information for Deep Neural Network in Fighting Game,” in *Proc. 9th International Conference on Agents and Artificial Intelligence (ICAART 2017)*, pp. 676-680, 2017.

[16] M. Ishihara, T. Miyazaki, Y. Chu, T. Harada, R. Thawonmas, “Applying and Improving Monte-Carlo Tree Search in a Fighting Game AI,” in *Proc. 13th International Conference on Advances in Computer Entertainment Technology*, ACM, no. 27, 2016.

[17] T. Kristo, N.U. Maulidevi, “Deduction of fighting game countermeasures using Neuroevolution of Augmenting Topologies,” in *Proc. 2016 International Conference on Data and Software Engineering (ICoDSE)*, 2016. DOI:10.1109/ICODSE.2016.7936127

[18] S. Yoshida, M. Ishihara, T. Miyazaki, Y. Nakagawa, T. Harada and R. Thawonmas, “Application of Monte-Carlo Tree Search in a Fighting Game AI,” in *Proc. IEEE 5th Global Conference on Consumer Electronics (GCCE)*, pp. 623-624, 2016.

[19] K. Majchrzak, J. Quadflieg, and G. Rudolph, “Advanced Dynamic Scripting for Fighting Game AI,” in *Proc. Entertainment Computing (ICEC 2015)*, pp. 86-99, 2015.

[20] K. Asayama, K. Moriyama, K. Fukui, and M. Numao, “Prediction as Faster Perception in a Real-time Fighting Video Game,” in *Proc. 2015 IEEE Conference on Computational Intelligence and Games (CIG 2015)*, pp. 517-522, 2015.

[21] N. Sato, S. Temsiririkkul, S. Sone. and K. Ikeda, “Adaptive Fighting Game Computer Player by Switching Multiple Rule-based Controllers,” in *Proc. 3rd International Conference on Applied Computing and Information Technology (ACIT 2015)*, pp. 52-59, 2015.

[22] H. Park and K.J. Kim, “Learning to Play Fighting Game using Massive Play Data,” in *Proc. 2014 IEEE Conference on Computational Intelligence and Games (CIG 2014)*, pp. 458-459, 2014.

[23] M. Ishihara, S. Ito, R. Ishii, T. Harada and R. Thawonmas, “Monte-Carlo Tree Search for Implementation of Dynamic Difficulty Adjustment Fighting Game AIs Having Believable Behaviors,” in *Proc. 2018 IEEE Conference on Computational Intelligence and Games (CIG 2018)*, 2018.

[24] S. Devlin, A. Anspoka, N. Sephton, P.I. Cowling, “Combining Gameplay Data with Monte Carlo Tree Search to Emulate Human Play,” in *Proc. Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE 2016)*, pp. 16-22, 2016.

[25] A. Zook and M.O. Riedl, “Learning How Design Choices Impact Gameplay Behavior,” *IEEE Transactions on Games*, Mar, 2018. (Early Access)

General Win Prediction from Agent Experience

Raluca D. Gaina
Queen Mary University of London
London, UK
r.d.gaina@qmul.ac.uk

Simon M. Lucas
Queen Mary University of London
London, UK
simon.lucas@qmul.ac.uk

Diego Perez-Liebana
Queen Mary University of London
London, UK
diego.perez@qmul.ac.uk

***Abstract*—The question of whether the correct algorithm is used for the problem at hand usually comes at the end of execution, when the algorithm's ability to solve the problem (or not) can be verified. But what if this question could be answered in advance, with enough notice to make changes in the approach in order for it to be more successful? This paper proposes a general agent performance prediction system, tested in real time within the context of the General Video Game AI framework. It is solely based on agent features, therefore removing potential human bias produced by game-based features observed in known games. Three different models can be queried while playing the game to determine whether the agent will win or lose, based on the current game state: early, mid and late game feature models. The models are trained on 80 games in the framework and tested on 20 new games, for 14 variations of 3 different methods. Results are positive, indicating that there is great scope for predicting the outcome of any given game.**

***Index Terms*—general video game playing, rolling horizon evolution, monte carlo tree search, win prediction**

I. Introduction

Many researchers have approached the problem of General Video Game Playing in the last years, partly encouraged by the proliferation of benchmarks like the General Video Game AI (GVGAI; [1]) framework, the Arcade Learning Environment (ALE; [2]) and others. In the different studies, authors employ either a single or a combination of techniques in order to tackle this complex problem, in which an agent must be able to play any within a collection of games. Known examples are Mnih *et al.* [3] work on the Atari framework or the several times winner of the GVGAI competition, YOLOBOT [4], respectively.

A recent survey in the uses of GVGAI for research and education [5] highlights a commonality on multiple studies: not only most approaches rarely surpass 25% of victories across the set of games tested (with just some approaches reaching 50% in particular games and settings), but also the victory count is accumulated in specific games. Some games in the framework are too hard for any of the GVGAI agents developed so far, and it remains an open question as to whether they are too hard for a general approach.

This is something that may be expected a priori, but as D. Ashlock *et al.* [6] suggest in their work, a hyper-heuristic approach or a portfolio of agents should be able to overcome this problem. However, finding the right approach for the right game, especially if that game is unknown, is a hard challenge.

Several previous studies suggest clustering games using game features or performance of agents on them [6], [7], [8]. In general, this clustering can be used to select which agent, from a collection of different techniques, should be used to play the game at stake. This is a reasonable approach, but little thought has been put so far into analysing if the algorithm should be changed once the game has already started. The technique used in a particular game may need to be discarded in favour of another one, either because the choice was wrong in the first place, or because the game conditions have changed.

In fact, it is common for a human who is playing a game to have a certain intuition about how well are they doing mid way through it. A player in Space Invaders can see, before losing the game, that the presence of too many aliens close to the ground is a bad sign. Having most pellets still to be eaten in Pac-Man with no power pills left in the level can also be an indication of the likely (negative) outcome of the game. Our interest is to see if it is possible to give this ability to a general agent, allowing the possibility of changing technique before it is too late in the game.

The use of game features, however, poses an additional problem: including the number of pills or aliens as features is a very specific approach. In fact, even considering GVGAI terms (as presence of Non-Player Characters, portals, resources, etc.) is not general enough. This does not only tailor the methods to GVGAI (which may be hard to avoid when working with a specific framework), but also to the games the algorithm designer has seen in the past. Other features, however, can be more resilient to this bias, such as agent-based features [9]: decisiveness of action selection, speed of convergence to a recommendation or analysis of the fitness landscape.

The work presented in this paper explores the idea of designing a game outcome predictor. In particular, we propose building predictors that only focus on agent-based features, in order not to bias the prediction towards already seen games. The question this paper tries to answer is if it is possible to train a model solely on agent experiences, so it is able to estimate the probability of victory at the current state for any game within the GVGAI framework.

II. Related work

There is extensive literature on extracting various AI gameplay measures. Traditionally, these methods are predominantly used in the area of Procedural Content Generation in order to assess the quality of a level or game created automatically.

Liapis *et al.* [10] create models of player types called "procedural personas", which then they use to automatically generate levels of a roguelike puzzle game. For this purpose,

978-1-5386-4359-4/18/$31.00 ©2018 IEEE

they identify several features that the evolved agents will focus on: the number of monsters they kill, the number of treasures collected or reaching the exit of the level. Using these different personas to automatically play-test levels, the authors are able to generate interesting levels which highlight agent strengths.

Some researchers focus more on the area of human-computer interaction and how measures extracted from game-play can be used in predicting various aspects characterizing automatically generated games (engagement, frustration and challenge in [11]; or human enjoyment when playing against different ghost teams in the game Ms Pac-Man [12]). The content and game-play features highlighted by Shaker *et al.* [11] in the platformer game Super Mario Bros are directly applicable to AI game-play as well as humans: number of enemies, number and width of gaps in the level, enemies placement, boxes, power-ups and events triggered during play.

Isaksen *et al.* [13] define several metrics characterising dice games: win bias (the difference between the probabilities of player A winning a dice battle and player A losing the battle), tie percentage (the probability of a tie in a given battle) and closeness (how much the result of the battle centered around a tie). Volz *et al.* [14] evaluate how close the game ended as well in the card game Top Trumps with the objective of automatically balancing the game.

However, the features explored by these authors are game-specific and applying these methods to other domains is not straightforward. When designing games, Browne and Maire [15] looked at 57 different criteria in judging an evolved game split into 3 categories, intrinsic, viability and quality. The authors use general game-playing agents to test their games, which are written in the Ludi Game Description Language. Most of the quality features analysed are resultant from AI game-play, such as depth, drama, decisiveness or uncertainty. Some of these metrics, where possible to translate to single-player games, were adapted for our study.

Several works move away from the area of PCG and instead focus on extracting measures of player behaviour to specifically tune game-playing agents or perform a deeper performance analysis than the typical win rate investigation. Khalifa *et al.* [16] used features from human game-play data to tune a human-like Monte Carlo Tree Search (MCTS) player. Their features mostly focused on actions, such as action repetition, change frequency or pauses, with an additional map exploration metric. The authors applied the features extracted from human data to tune a Monte Carlo Tree Search agent on 3 different games in the General Video Game AI framework (GVGAI), with mixed results.

More general measures for better analysis are depicted by Volz *et al.* in [9]. Their prototype implementing the measures for live game-playing agent analysis also uses the GVGAI framework, allowing for a general application of the method on several different games. Some of these metrics, such as decisiveness or action entropy, were included in this study, excluding multi-player or comparison metrics.

Some researchers use such metrics for machine learning tasks. For example, Bontrager *et al.* [7] cluster the games in the GVGAI framework based on the performance of several agents submitted to the corresponding competition. In this case, the performance of an agent is simply characterized by the win rate, which is shown to differ between the players. The authors signify that some agents possess skills useful in certain tasks, while other agents lack or make up for them in different ways.

Mendes *et al.* [8] used this conclusion to construct a hyper-heuristic agent. The authors extracted several game features (number and type of NPCs, resources available, map dimensions and number and types of other sprites) and used a classification method to determine which AI agents, selected from a subset of GVGAI competition entries, achieve highest win rates when specific game features or combination of features are present in a new game tested. The algorithm then decides which agent to query for a solution depending on the recommendation of the classifier (a Support Vector Machine and a Decision Tree). The agent selected will play the entire game with no changes.

A similar approach was employed by Horn *et al.* in [17] for AI hybrid evaluation (excluding the hyper-heuristic construction step). They propose a game difficulty estimation scheme based on game features (NPC types, puzzle elements, pathfinding requirements or traps). These are arguably more open to human bias, as each metric is evaluated manually. Although the game difficulty features identified do not correspond to agent win rates, the authors carry out an analysis which gives a deeper insight into reasons for agent performance levels.

These works are, however, based on game features as defined by human knowledge on the existing data set. This paper proposes a game win predictor based solely on agent experiences, aiming to remove potential human bias resultant from designing features seen on known games.

III. Background

A. General Video Game AI

The domain chosen for this study is the General Video Game AI (GVGAI) framework [18], which allows for general video-game playing agent testing on a wide range of different games. The diversity in game features, difficulty and different agent performance is showcased in the previous section, which highlights it as an appropriate environment for general agent testing. As opposed to other frameworks focused on the area of General (Video) Game Playing, GVGAI does not make the game ruleset available to the AI agents. Instead, the information given contains the current game state with the sprites present in the level, the avatar the agent is controlling, the action set available, the history of events up until that game tick and the current game score. Additionally, in the planning tracks, agents have access to a non-deterministic Forward Model (FM), which they may use to simulate possible future states. All information is offered to the agents through Java objects. All games in the framework are real-time, giving 1 second initialization time and 40 milliseconds decision time at every game tick. There are 100 single-player grid-physics games available in the framework as of March 2018, all of which are used in this study for a large scale experiment.

B. Game-playing agents

This section describes the 3 methods that the game-playing agents used in this study are based on.

1) Random Search (RS): This agent uniformly samples at random action sequences of length L within the allocated budget and chooses for play the first action in the best solution found. In order to evaluate a sequence, the FM is used to simulate through the actions, in turn, until the end of the game or the end of the sequence is reached. The value of the final state is computed using a heuristic (see Equation 1, where H^+ is a large positive integer number and H^- is a large negative integer number), this becoming the value of the solution.

$$f = score \; + \; \begin{cases} H^+, \; if \; loss \\ H^-, \; if \; win \end{cases} \tag{1}$$

2) Rolling Horizon Evolutionary Algorithm (RHEA): This agent is one of the promising methods in the domain of General Video Game Playing, as showcased in [19]. In its vanilla form, it randomly initializes a population of size P with individuals of length L, which it then evolves over several generations by applying various evolutionary techniques, such as uniform crossover and uniform 1-bit mutation. One individual represents a sequence of actions, which is evaluated similarly to the RS procedure: the FM model is used to simulate through the sequence of actions and the final state is evaluated with the same heuristic described in Equation 1. The first action of the best individual found at the end of the evolution is selected.

Previous work is used to select the agents employed in this study, the best of each being chosen. As a result, vanilla RHEA [20], EA-MCTS [21] and EA-Shift [19] form the subset of RHEA variations. EA-MCTS employs a different seeding method, using the solution provided by a Monte Carlo Tree Search agent (awarded half the thinking budget) to generate its initial population. EA-Shift focuses on Monte Carlo roll-outs added at the end of individual evaluation, as well as a shift buffer applied for population management (the population is not discarded and reinitialized at every game tick, but instead shifted to the left and new random actions are added at the end of each individual). Additionally, we add EA-All for completeness, which combines EA-Shift with EA-MCTS.

3) Monte Carlo Tree Search (MCTS): This agent is the most dominant method in GVGAI, many competitors choosing it as the basis for their entry. A comprehensive survey of MCTS techniques can be found in [22]. MCTS builds an asymmetric tree to make its choices, relying on statistics gathered from several simulated play-throughs. Each iteration that adds to the tree statistics begins by navigating down the tree using the tree policy (Upper Confidence Bound for Trees with an exploration constant of $\sqrt{2}$ for the agents used in this study, aiming to balance between exploration and exploitation). When it finds a node not yet fully expanded, a new child of this node is added to the tree, by selecting a new action to play from this game state. A Monte Carlo simulation (or roll-out) is run from the newly added child until the end of the game or a depth L is reached. The final state is evaluated with the same heuristic from Equation 1 and the value is backed up the tree, updating all nodes visited during this iteration. In our implementation, the nodes only store statistics and not the actual game states, the FM being used to simulate through the tree at every step. W is the number of iterations used for analysis. The most visited action at the end of the process is selected for play.

C. Classification

Due to the high variety of the games in the GVGAI framework and the low overall performance of the general agents (most games remain too difficult to be solved), as highlighted in the literature review, the $F1$-Score (see Equation 2) will be reported as to the quality of the classifiers employed in this study. It represents the harmonic average between precision and recall, 1 signifying the best value and 0 the worst.

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{2}$$

This is meant to be a better measure of classifier quality than accuracy when there is an imbalance in data (in this case, a majority of games resulting in a loss, see Table I) [23].

IV. Data set

To obtain the set of agents used to generate the data set, 3 roll-out values L were tested for RS (10, 30, 90); 2 parameter sets were tested for all 4 RHEA variations (P=2,L=8 and P=10,L=14); 3 parameter sets were tested for MCTS (W=2,L=8; W=10,L=10 and W=10,L=14). Every experimental setup makes use of the same number of FM calls.

All 14 algorithm variations described previously were run on all 100 games publicly available in the GVGAI Framework, 20 times on each of the 5 levels, being given a budget of 900 FM calls. Each run produced 2 log files, recording information about the agent inner processing, as well as its actions played and game scores, at every game step, in addition to the final game results (win/loss, final score and number of game ticks).

Data set and processing scripts are publicly available[1]. On each game, Formula-1[2] points are awarded attending to a ranking determined by win rate. The first 10 ranked entries receive 25 points, second 18, then 15, 12, 10, 8, 6, 4, 2, 1 and 0 for the 11^{th} and below positions. Points across games are summed up for an overall ranking, shown in Table I.

A list of all the features extracted can be found below. Features ϕ_2, ϕ_8, ϕ_9, ϕ_{10}, ϕ_{11} and ϕ_{12} compute averages from the beginning of the game up until the current tick t. Features ϕ_5, ϕ_6, ϕ_{11} and ϕ_{12} rely on the FM. Only *agent features* were used in this study, with the exception of the game score:

ϕ_1 **Current game score**

ϕ_2 **Convergence**: Iteration number when the algorithm found the final solution recommended during one tick. A low value indicates quick and almost random decisions.

ϕ_3 **Positive rewards**: Count of positive scoring events.

ϕ_4 **Negative rewards**: Count of negative scoring events.

[1] https://github.com/rdgain/ExperimentData/tree/GeneralWinPred-CIG-18

[2] Not to be mistaken with F1 accuracy measure for classifiers.

#	Algorithm	Points	Avg. Wins
1	**10-14-EA-Shift**	1225	26.02 (2.11)
2	**10-RS**	898	24.33 (2.13)
3	**2-8-EA-All**	888	23.95 (1.98)
4	**30-RS**	885	22.49 (2.02)
5	**2-8-EA-Shift**	866	24.54 (2.00)
6	**14-MCTS**	780	24.29 (1.74)
7	**10-14-EA-All**	695	22.66 (2.02)
8	**10-14-RHEA**	664	23.23 (2.08)
9	**10-MCTS**	652	24.01 (1.65)
10	**2-8-EA-MCTS**	621	23.98 (1.73)
11	**10-14-EA-MCTS**	618	23.99 (1.80)
12	**8-MCTS**	594	23.42 (1.61)
13	**90-RS**	457	16.31 (1.67)
14	**2-8-RHEA**	257	18.33 (1.77)

TABLE I: GVGAI-style Formula-1 point ranking of all methods. Type and configuration (roll-out length L if one value, population size P and roll-out length L if two values) are reported, followed by the sum of Formula-1 points across 20 games and the average win rate.

ϕ_5 **Success**: The slope of a line over all the win counts. Win count increases when any solution sees the end of the game with a win, at any point during search. A high value indicates the increase in discovery of winning states.

ϕ_6 **Danger**: The slope of a line over all the loss counts. Loss count increases when any solution sees the end of the game with a loss, at any point during search. A high value indicates the increase in discovery of losing states.

ϕ_7 **Improvement**: The slope of a line resultant from all best fitness values plotted over game tick. A high value indicates good fitness improvement.

ϕ_8 **Decisiveness**: Shannon entropy (SE) over the number of times each of the possible actions was recommended (it was the first action of a solution in the final population or analysis window). In all cases of distribution-based features, a high value suggests actions of similar value; the opposite shows some to be dominating.

ϕ_9 **Options exploration**: SE over the number of times each of the possible actions was explored (it was the first action of a solution at any time during search). A low value shows an imbalance in actions explored.

ϕ_{10} **Fitness distribution**: SE over fitness per action.

ϕ_{11} **Success distribution**: SE over win count per action.

ϕ_{12} **Danger distribution**: SE over loss count per action.

The full feature file (processing all games and algorithms for global classifiers) took approximately 2.5 hours to generate, from 26GB of raw metrics data split over 281.4k files (Dell Windows 10 PC, 3.4 GHz, Intel Core i7, 16GB RAM, 4 cores).

Figure 1 shows the pairwise correlation between the features extracted (using the Pearson correlation coefficient), in a comparison between the early (first 30% of game ticks) and late (last 30% of game ticks) phases of the games. Differences are small, but they do exist. An aspect worth highlighting is the higher correlations in the bottom right corner in the late game phase versus the early game phase (i.e. the success distribution appears to increase correlation with all other features).

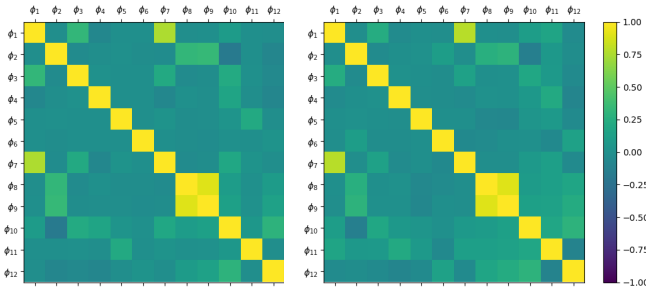


Fig. 1: Feature correlation early game (left, 0-30% of all games) and late game (right, 70-100% of all games)

Another interesting positive correlation that only appears in the late game phase is that between the sense of danger and the convergence, suggesting agents take longer to settle on their final decision when surrounded by possible losses. The case of one action appearing to be dominating leads to a persistent negative correlation between convergence and fitness distribution. This suggests that agents are unlikely to change their decision if one action is deemed significantly better than the rest and try a less promising move.

V. Predictive Models

For the purpose of these experiments, all data sets were randomly split 80/20 in training/test subsets. This study aims to build several classifier models from agent features extracted, which would predict a win or a loss during play of a new game. We also show that the system is robust enough to handle new agents with significantly different play styles as well.

It takes approximately 10 seconds to process a full feature file and split the data into train and test, another 10 seconds to train a global model on a full feature file (or 1 minute if cross validation is used). Predicting the outcome of 28000 instances takes approximately 1 minute, the equivalent of 2.26ms per instance. As the data used in this study is publicly available, adapting the methods to different problems or agents would only involve extracting the relevant features from the newly introduced agents or problems.

A. Baseline

The baseline model all our classifiers are compared against is a simple rule based predictor incorporating human knowledge. In classic arcade games and most GVGAI games, gaining score is a good thing and often means the player is on the right path to winning if they increase their score. This idea is implemented as described in Equation 3, which compares the count of positive scoring events recorded to the count of negative events. This classifier's performance on the test set is shown in Table II, where it can be observed that it reaches an F1-Score of only 0.59 despite a high precision (0.70). This model will be referred to as R_g in the rest of this paper.

$$\hat{y} = \begin{cases} win & if\ \phi_3 > \phi_4 \\ lose & otherwise \end{cases} \quad (3)$$

	Precision	Recall	F1-Score	Support
Loss	0.83	0.52	0.64	20500
Win	0.35	0.70	0.46	7500
Avg / Total	**0.70**	**0.57**	**0.59**	**28000**

TABLE II: Global rule based classifier report. Global model tested on all game ticks of all instances in the test set.

	Precision	Recall	F1-Score	Support
Loss	1.00	0.99	0.99	20500
Win	0.97	0.99	0.98	7500
Avg / Total	**0.99**	**0.99**	**0.99**	**28000**

TABLE III: Global AdaBoost classifier report. Global model tested on all game ticks of all instances in the test set.

ϕ_1	0.24	ϕ_2	0.04	ϕ_3	0.08	ϕ_4	0.06
ϕ_5	0.2	ϕ_6	0.1	ϕ_7	0.12	ϕ_8	0
ϕ_9	0.06	ϕ_{10}	0.02	ϕ_{11}	0.02	ϕ_{12}	0.06

TABLE IV: Feature importances extracted from global model. ϕ_x represents a feature and its associated importance.

B. Classifier selection - global model

Seven classifiers (with default hyper-parameters if not specified) were trained and tested for proof of concept and classifier analysis. These are K-Nearest Neighbors (5 neighbours), Decision Tree (5 max depth), Random Forest (5 max depth, 10 estimators), Multi-layer perceptron (learning rate 1), AdaBoost-SAMME [24], Naive Bayes and Dummy (simple rule decision making, very poor general performance to be used as another possible baseline). All classifiers use the implementation in the Scikit-Learn Python 2.7.14 library [25].

Cross-validation with 10 folds was used during training to assess performance, the classifiers obtaining 0.95, 1.00, 0.98, 0.96, 1.00, 0.95 and 0.66 accuracy during validation, respectively. Both AdaBoost and the Decision Tree classifier achieved high accuracy values during validation and test (0.99, see Table III for its performance measures) and were deemed equal. Either could be used, but AdaBoost was selected as the main classifier for the rest of the experiments presented.

Feature importances according to AdaBoost can be seen in Table IV. It appears that the game score is most important in distinguishing wins and losses, unsurprisingly, but it is followed close behind by the number of wins seen by the agents, the improvement in fitness and the sense of danger. The decisiveness of the agents is considered to have no impact in deciding the outcome of a game.

C. Model training

All games were split into logical phases for predictions at various points in the games: early game ($0-30\%$), mid game ($30-70\%$) and late game ($70-100\%$). Multiple models were then trained for each of the phases, using agent features based on metrics logged only in the ticks corresponding to each interval. 3 different models resulted, referred to as E_g, M_g and L_g, respectively, in the rest of this paper.

The performance of the models was analysed by testing each on the 20 new games, on their corresponding interval of game ticks. During training with 10-fold cross-validation, they achieve 0.80, 0.82 and 0.99 accuracy, respectively. During test on the new games, they report accuracies of 0.73, 0.80 and 0.99 (0.70, 0.80 and 0.99 F1-Scores), respectively. These results are satisfactory and allow for further exploration.

VI. Live play results

For the experiments in this paper, we simulated live play by extracting agent features from the log files for a range of ticks ($T = \{100 \cdot a : \forall\ a \in [1, 20] : a \in \mathbb{N}\}$), all from the beginning of the game until the current tick tested $t \in T$. Gameplay from all 14 algorithms on the 20 test games (20 plays on each of the 5 levels) was used to compute the final results. Each model was tested on each of the feature files, being asked to predict the game outcome every 100 ticks.

Simulated live play results can be observed in Figure 4. The simple rule based model achieves a high performance in some of the games and it proves better than the trained predictive models (i.e. "Aliens", "Defem", "Chopper", "Eggomania"). As these are games with plenty of scoring events, it is unsurprising that the simple logic of R_g works in these cases. However, there are games where the trained models achieve much better predictions ("Ghost Buster", "Colour Escape" or "Frogs"). The reward gain is not linear in these games, meaning the player need not necessarily be phased by a temporary decrease in score, or too optimistic as a result of score gains.

It is interesting to observe that the trained models do not follow the expected curves (E_g being better in the early game phase and then decreasing, M_g showing a spike in the middle of the game and L_g offering good predictions only towards the end of the game). Instead, the early game model appears to have a generally low performance compared to the rest, which can be explained by the limited information available for this particular model. The late game model seems particularly strong in games with very low win rate ("Fireman", for example, in which both E_g and M_g are predicting wins, yet the overall win rate remains at 0% for this game).

It is most interesting to observe the games with close to 50% win rate, "Defem" and "Ghost Buster". High F1-Score values here indicate that the predictors are able to correctly judge both wins and losses equally. And indeed, in both games, the trained models achieve F1-Scores of over 0.8 only half way through the game. Model M_g appears to excel in these situations, meaning that it can recommend the game outcome and possibly the better approach to be used.

It is important to highlight at this point the importance of this great result: the predictor is able to foresee with high reliability, after only a fourth of the game has been played, if the agent is going to win or lose the game. In this case, games that are either won or lost with the same probability as a coin flip. And these are truly general models: trained in *different* games, using only *agent experience* features. This shows a great scope for the system's use within hyper-heuristic methods, as some of the algorithms tested in this study do win at "Defem" and "Ghost Buster". Devising a procedure that determines which is that better method and switches to

	Early-P	Mid-P	Late-P	Total-M
E_g	0.22 (0.72)	0.42 (0.74)	0.49 (0.76)	0.38 (0.74)
M_g	0.29 (0.72)	**0.57 (0.79)**	**0.71 (0.83)**	**0.53 (0.78)**
L_g	0.01 (0.73)	0.05 (0.74)	0.22 (0.76)	0.09 (0.74)
R_g	**0.42 (0.67)**	0.47 (0.61)	0.46 (0.58)	0.45 (0.62)
Total-P	0.24 (0.71)	0.38 (0.72)	**0.47 (0.73)**	

TABLE V: F1-Scores each model per game phase over all games, accuracy in brackets. Each row is a model, each column is a game phase. Highlighted in bold is the best model on each game phase, as well as overall best phase and model.

it when the prediction is a loss is scope for future work, but having a system that indicates if a change should be made is the first step in that direction.

All predictive models were further analysed as to their average quality considering all games. To this extent, table V summarises F1-scores for all models on the different game phases identified. The models are the same as discussed in Section V-C, and they are tested in the same previous test setting, with features extracted from the beginning of the game until the current tick which falls at the half point in each game phase (15%, 50% or 85% of the game ticks).

The results indicate the rule-based model to be giving consistent average performance throughout the game phases, being the best in the early phase with an F1-score of 0.42. In the Mid and Late game phases, model M_g is the best, achieving a 0.57 and 0.71 F1-score, respectively. Overall, the best model is M_g with an F1-score average of 0.53.

It is not surprising that the M_g model is the best in its respective game phase, and it is expected that the prediction quality is generally lower in the Early game phase, when there is less information available and it is harder to judge if the agent's performance is good enough or not. A significant result extracted from the summarised data is that model M_g achieves high (if not the best) F1-scores across all game phases, indicating that the system can identify with high confidence whether the agent is performing well or not and leaving open the possibility of switching approaches appropriately.

VII. Conclusion

This paper presents work in extracting agent features from AI gameplay in a generic setting, using the General Video Game AI framework (GVGAI). Game-specific features are specifically excluded in order to avoid potential bias introduced by human knowledge of already known games. 14 total variations of Rolling Horizon Evolutionary Algorithm, Monte Carlo Tree Search and Random Search are used to generate data on 100 games, playing 20 times each of the 5 levels. Three different models corresponding to early, middle (mid) and late game phases are trained on 80 randomly selected games and tested on the remaining 20 through live play simulation and repeated predictions every 100 game ticks.

The results obtained indicate that models are able to correctly predict in most cases the outcome of the game with sufficient time before the end of the game to make appropriate changes in the method employed. Throughout all experiments, it is apparent that some models have better predictions in specific games than others. Additionally, the mid-game phase model proved to have the best overall performance, achieving an F1-Score of 0.53 (0.78 accuracy) across all test games and game phases. It is also the strongest model in the individual mid and late phases, being bested in the early game phase only by the simple rule predictor implemented (which incorporates the human knowledge that gaining score leads to a win).

Regarding next steps, a hyper-heuristic agent will be built, able to switch between algorithms appropriately while playing the game, based on the predictions given by our system. The task can be split into two: identifying which features need improvement and which method leads to the desired behaviour. A prediction explanatory system could be responsible for the first part of this task and first steps towards this system are presented in Figure 2, which uses the LIME system[3]. The example provided is an explanation of the prediction of each model at game tick 300 in "Frogs" level 0, when played by 2-8-RHEA. There is an obvious difference between features and a clear signaling of which features currently indicate a loss. Therefore, a hyper-heuristic method could make use of this analysis to correct the loss indications.

Additionally, new methods could be introduced to the system in order to create stronger models, able to adapt to any style of play. The current system is robust enough to handle testing on new algorithms: Figure 3 shows predictions trained with data generated only by RHEA and RS variants, but tested live with an MCTS controller playing the game. If this is compared to Figure 4s, it can be seen that all models are able to maintain a similar shape and still accurately predict the outcome half way through the game.

Lastly, more features could be integrated to better describe player experience, such as empowerment [26], spatial entropy or characterization of agent surroundings [9].

Acknowledgment

This work was funded by the EPSRC Centre for Doctoral Training in Intelligent Games and Game Intelligence (IGGI) EP/L015846/1.

References

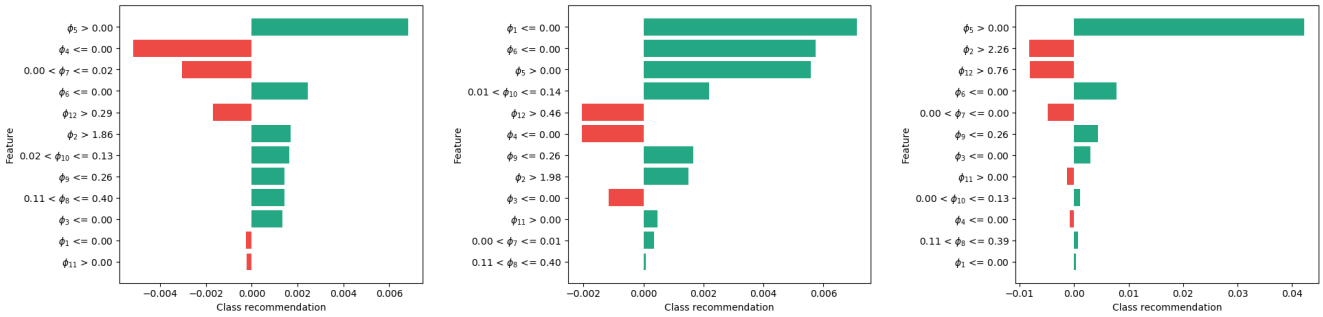
[1] D. Perez-Liebana, S. Samothrakis, J. Togelius, S. M. Lucas, and T. Schaul, "General Video Game AI: Competition, Challenges and Opportunities," in *Thirtieth AAAI Conference on Artificial Intelligence*, 2016, pp. 4335–4337.

[2] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents." *J. Artif. Intell. Res.(JAIR)*, vol. 47, pp. 253–279, 2013.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[4] T. Joppen, M. Moneke, N. Schroder, C. Wirth, and J. Furnkranz, "Informed Hybrid Game Tree Search for General Video Game Playing," *IEEE Transactions on Computational Intelligence and AI in Games*, 2017.

[3]https://github.com/marcotcr/lime



(a) Early model (b) Mid model (c) Late model

Fig. 2: Class predictions by features. Red signifies the model feature recommends a loss, green a win. The probability of class being selected based on individual feature recommendation is plotted on the X-axis.

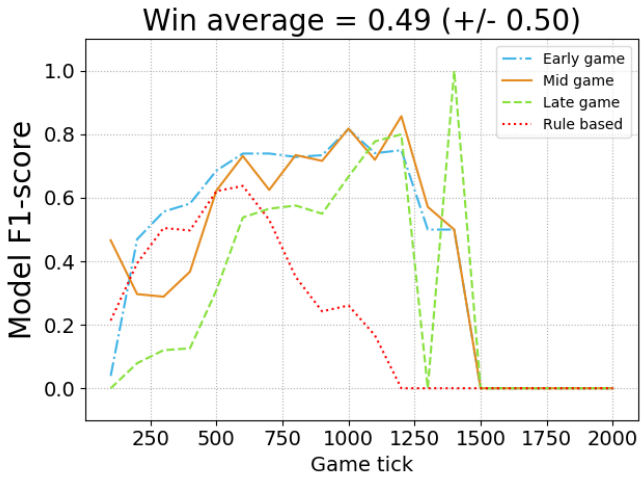


Fig. 3: Model F1-scores in the game `Ghost Buster`, trained on variations of RHEA and RS (80 training games) and tested on MCTS.

[5] D. Perez-Liebana, J. Liu, A. Khalifa, R. D. Gaina, J. Togelius, and S. M. Lucas, "General video game ai: a multi-track framework for evaluating agents, games and content generation algorithms," *arXiv preprint arXiv:1802.10363*, 2018.

[6] D. Ashlock, D. Pérez-Liébana, and A. Saunders, "General Video Game Playing Escapes the No Free Lunch Theorem," in *2017 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2017.

[7] P. Bontrager, A. Khalifa, A. Mendes, and J. Togelius, "Matching games and algorithms for general video game playing," in *Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2016, pp. 122–128.

[8] A. Mendes, J. Togelius, and A. Nealen, "Hyper-heuristic general video game playing," in *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, Sept 2016, pp. 1–8.

[9] V. Volz, D. Ashlock, S. Colton, S. Dahlskog, J. Liu, S. M. Lucas, D. P. Liebana, and T. Thompson, "Gameplay Evaluation Measures," in *Articial and Computational Intelligence in Games: AI-Driven Game Design (Dagstuhl Seminar 17471)*, E. André, M. Cook, M. Preuß, and P. Spronck, Eds. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018, pp. 36–39.

[10] A. Liapis, C. Holmgård, G. N. Yannakakis, and J. Togelius, "Procedural personas as critics for dungeon generation," in *European Conference on the Applications of Evolutionary Computation*. Springer, 2015, pp. 331–343.

[11] N. Shaker, S. Asteriadis, G. N. Yannakakis, and K. Karpouzis, "Fusing visual and behavioral cues for modeling user experience in games," *IEEE Transactions on Cybernetics*, vol. 43, no. 6, pp. 1519–1531, Dec 2013.

[12] W. Sombat, P. Rohlfshagen, and S. M. Lucas, "Evaluating the enjoyability of the ghosts in ms pac-man," in *2012 IEEE Conference on Computational Intelligence and Games (CIG)*, Sept 2012, pp. 379–387.

[13] A. Isaksen, C. Holmgård, J. Togelius, and A. Nealen, "Characterising score distributions in dice games," *Game and Puzzle Design*, vol. 2, no. 1, 2016.

[14] V. Volz, G. Rudolph, and B. Naujoks, "Demonstrating the feasibility of automatic game balancing," *CoRR*, vol. abs/1603.03795, 2016. [Online]. Available: http://arxiv.org/abs/1603.03795

[15] C. Browne and F. Maire, "Evolutionary game design," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 2, no. 1, pp. 1–16, 2010.

[16] A. Khalifa, A. Isaksen, J. Togelius, and A. Nealen, "Modifying mcts for human-like general video game playing." in *IJCAI*, 2016, pp. 2514–2520.

[17] H. Horn, V. Volz, D. Prez-Libana, and M. Preuss, "MCTS/EA hybrid GVGAI players and game difficulty estimation," in *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, Sept 2016, pp. 1–8.

[18] D. Perez-Liebana, S. Samothrakis, J. Togelius, T. Schaul, S. Lucas, A. Couetoux, J. Lee, C.-U. Lim, and T. Thompson, "The 2014 General Video Game Playing Competition," in *IEEE Transactions on Computational Intelligence and AI in Games*, vol. PP, no. 99, 2015, p. 1.

[19] R. D. Gaina, S. M. Lucas, and D. P. Liébana, "Rolling Horizon Evolution Enhancements in General Video Game Playing," in *Proceedings of IEEE Conference on Computational Intelligence and Games*, 2017.

[20] R. D. Gaina, J. Liu, S. M. Lucas, and D. P. Liébana, "Analysis of Vanilla Rolling Horizon Evolution Parameters in General Video Game Playing," in *Springer Lecture Notes in Computer Science, Applications of Evolutionary Computation, EvoApplications*, no. 10199, 2017, pp. 418–434.

[21] R. D. Gaina, S. M. Lucas, and D. P. Liébana, "Population Seeding Techniques for Rolling Horizon Evolution in General Video Game Playing," in *Proceedings of the Congress on Evolutionary Computation*, 2017.

[22] C. Browne, E. Powley, D. Whitehouse, S. Lucas, P. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A Survey of Monte Carlo Tree Search Methods," in *IEEE Trans. on Computational Intelligence and AI in Games*, vol. 4, no. 1, 2014, pp. 1–43.

[23] H. He and E. A. Garcia, "Learning from imbalanced data," *IEEE Transactions on knowledge and data engineering*, vol. 21, no. 9, pp. 1263–1284, 2009.

[24] J. Zhu, S. Rosset, H. Zou, and T. Hastie, "Multi-class AdaBoost," *Ann Arbor*, vol. 1001, no. 48109, p. 1612, 2006.

[25] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux, "API design for machine learning software: experiences from the scikit-learn project," in *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, 2013, pp. 108–122.

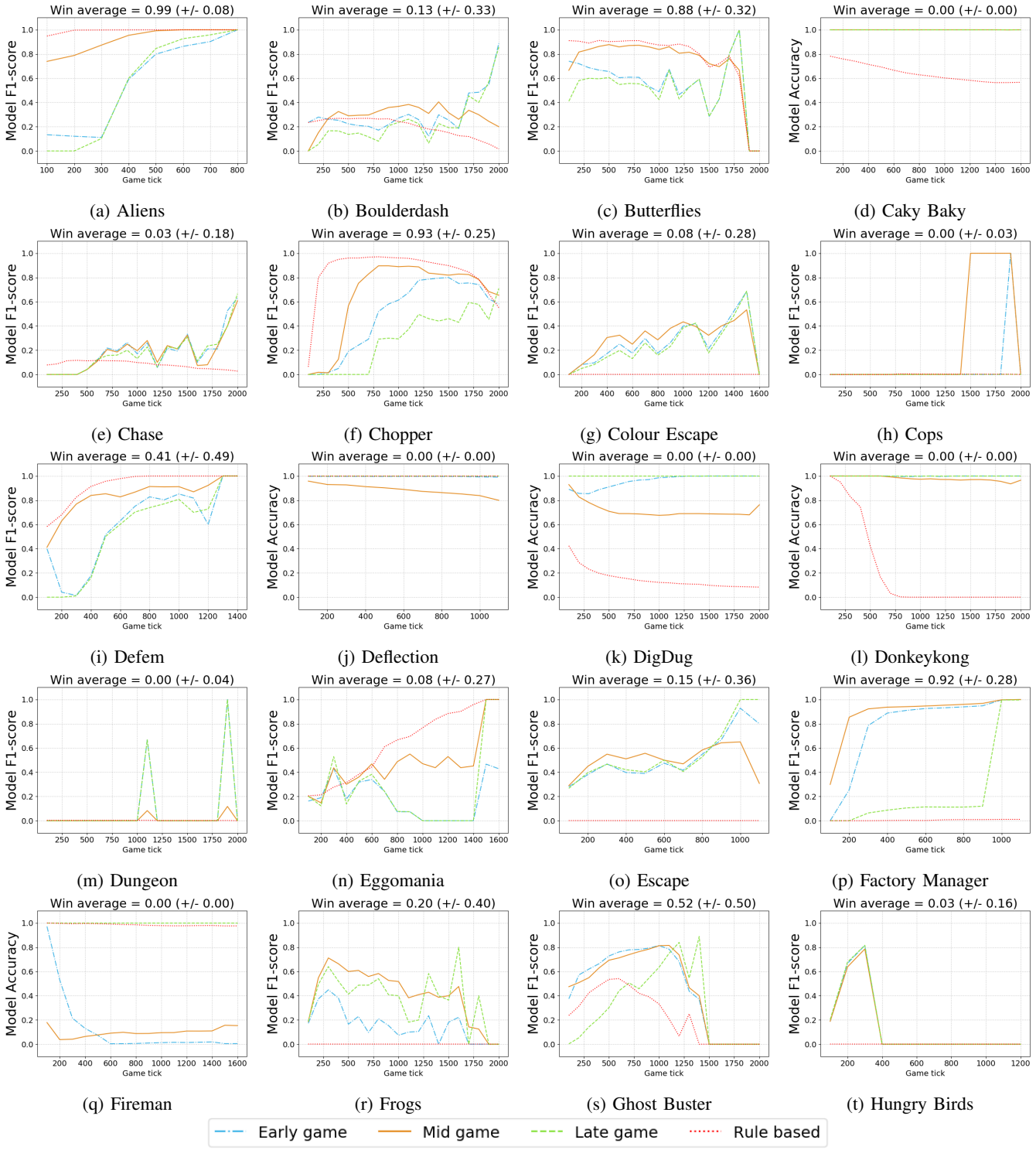


Fig. 4: Model F1-scores for each game in the test set, averaged over up to 1400 runs, 14 agents, 100 runs per game. Game ticks are displayed on the X axis, maximum 2000 game ticks. 3 different predictor models trained on early, mid and late game data features, as well as the baseline rule-based predictor. If F1-scores were 0 for all models, accuracy is plotted instead. Additionally, win average is reported for each game.

[26] C. Guckelsberger, C. Salge, J. Gow, and P. Cairns, "Predicting player experience without the player.: An exploratory study," in *Proceedings of the Annual Symposium on Computer-Human Interaction in Play*, ser. CHI PLAY '17. New York, NY, USA: ACM, 2017, pp. 305–315.

Building Evaluation Functions for Chess and Shogi with Uniformity Regularization Networks

Shanchuan Wan
Graduate School of Interdisciplinary Information Studies
The University of Tokyo
Tokyo, Japan
swan@graco.c.u-tokyo.ac.jp

Tomoyuki Kaneko
Interfaculty Initiative in Information Studies
The University of Tokyo, and
JST, PRESTO
kaneko@acm.org

***Abstract*—Building evaluation functions for chess variants is a challenging goal. At this time, only AlphaZero succeeded with millions of self-play records produced by using thousands of tensor processing units (TPUs), which are not available for most researchers. This paper presents the challenge of training evaluation functions on the basis of deep convolutional neural networks using decent data and computing resources, where regularization is crucial as complex models trained with limited data are more prone to overfitting. We present a novel training scheme by introducing a uniformity regularization (UR) network. In the proposed approach, a value network and a discriminator network share common convolutional layers and both networks are trained simultaneously. Loss functions for them are based on the difference between the score of a random move and that of an experts' move as a comparison training method. The value network is expected to give precise scores for all positions, while the discriminator makes qualitative evaluations for move pairs, and acts as a regularizer that penalizes differences in evaluation results to ensure all samples are uniformly discriminated. Due to the existence of shared layers, such regularization is beneficial for improving the overall accuracy of the value network. Experimental results for chess and shogi demonstrate the proposed method surpassed the standard L_2 regularization method, and successfully helped obtain decently accurate value networks.**

***Index Terms*—neural networks, joint training, regularization, computer chess**

I. Introduction

Chess is one of the most popular and classic intellectual games in the world. Along with its variants, chess has been the subject of focus for researchers for a long time. The match in which world chess champion Garry Kasparov was defeated by the chess-playing machine Deep Blue [1] is treated as a great milestone in the development history of computer science and artificial intelligence.

For modern chess programs, a game tree search algorithm and an evaluation function are two critical components. The search algorithm is for simulating the game process and determining the best moves, while the evaluation function is responsible for assessing the advantage of a given position when it appears as a node on the game tree. Generally, high-accuracy evaluation functions provide more reliable search results with less occurrences of mistaken decisions. Automated construction of evaluation functions without hand-crafted features has been one of the most challenging research goals.

Silver et al. [2] introduced AlphaZero, which could obtain accurate evaluation functions with deep convolutional neural networks (DCNNs) for chess and shogi (Japanese chess) via reinforcement learning, without giving preset domain knowledge. Although the results including playing strength were promising, its learning involved millions of self-play records produced by using thousands of tensor processing units (TPUs), which are not available for most researchers. Therefore, there is still the open question of how accurate evaluation functions can be obtained in a limited amount of data as well as with reasonable computational resources.

The purpose of this paper is to provide a training scheme toward achieving that goal with a novel regularization technique designed for DCNN-based value networks. With the proposed solution, relatively large value networks based on 32-layer residual convolutional layers were successfully trained with only two graphics processing units (GPUs) within two days without overfitting. This paper has two main contributions. The first is a new joint training scheme in which a value network and an assistant discriminator network share their weights in convolutional layers. Both networks are trained with the same data simultaneously. The other is a carefully crafted regularization term (acts as a regularizer) added to the overall objective function, which regulates the output of the newly introduced discriminator network and ensure that all samples are uniformly discriminated. Through shared convolutional layers, the effects of such regularization can be propagated to the main parts of the value network.

Through experiments in chess and shogi, the proposed uniformity regularization network was proven that it can be applied in conjunction with multiple objective functions of value networks and bring considerable improvements to them and can work much better than L_2 regularization [3] when training data are relatively sufficient. Also, within limited training time, value networks trained by the proposed method reached the frontier level of predicting expert moves.

II. Related Work

A. Learning of Evaluation Functions in Games

Silver et al. demonstrated via AlphaGo Zero and AlphaZero that accurate DCNN-based value networks could be trained in Go, chess, and shogi by very large-scale reinforcement

978-1-5386-4359-4/18/$31.00 ©2018 IEEE

learning [2], [4]. Their work has been considered remarkable in that only a minimal amount of domain knowledge was given and no hand-crafted features were used. In their work, value networks output the winning probability of a given position, which are equivalent to evaluation functions in traditional studies.

In this paper, we attempt to further show the effectiveness of DCNNs for value networks without using hand-crafted features on the basis of Silver et al.'s work. However, we do not adopt reinforcement learning due to resource limitations. Instead, we focus on how well we can train value networks given a set of game records.

There have been many studies on how to improve evaluation functions by using game records. In AlphaZero, value networks are trained to predict the winner of self-play games. Note that the winner of self-play games is equivalent to a Monte-Carlo return of episodes in reinforcement learning. Similarly, Buro successfully trained Logistello's evaluation function in Othello to predict the final score (the difference of disks) of a given position [5].

Temporal difference learning is also a major approach in reinforcement learning, as well as Monte-Carlo control methods. It learns evaluation functions by minimizing the differences between the score of current state and those of subsequent states. TD-LEAF(λ) is a variant of temporal difference learning applied to chess [6]. Giraffe [7] adopted TD-LEAF(λ) for training of its evaluation functions in chess implemented with neural networks. We compared our networks with Giraffe in experiments.

Another teaching signal is a move played. Comparison training [8] is a classic method to train a network to choose the best option among a set of alternatives. A discretized version of the comparison training was adopted to tune a subset of the weights in Deep Blue [9]. Minimax tree optimization (MMTO) method was developed based on comparison training to learn the heuristic evaluation function of search programs [10]. MMTO successfully tuned the full set of weights in the evaluation function of Bonanza, a famous shogi program that achieved first place at the World Computer Shogi Championship in 2006 and 2013. For maximizing the move decisions to agree with human experts, Bonanza applied a numerical iterative method to determine local minima for a specialized objective function, and more than 40 million parameters were adjusted. In this paper, we adopt a loss function based on comparison training as the objective to minimize in the training of value networks for evaluation functions.

DeepChess [11] also adopted neural networks for chess. Their model is a multi-layer perceptron network which takes two positions as the input and outputs that which is better. They also developed a modified $\alpha\beta$ search algorithm. However, it is different from typical value networks in other studies that assign a scalar value for a given position.

To the best of our knowledge, only AlphaZero and our work successfully trained evaluation functions in DCNNs, without hand-crafted features, for chess variants.

B. Stabilization of Training and Regularization

Despite deep neural networks are powerful function emulators, the training of such large models involves many issues have to be resolved including exploding gradient and overfitting problems.

The terms of "value network" and "policy network" used in this paper originated from AlphaGo [12]. A value network is trained to predict the winner of a position, and a policy network is trained to predict an expert move in supervised learning or self-play move in reinforcement learning. In the subsequent work of AlphaGo, AlphaGo Zero [4] adopted an integrated network with two sub-networks (value and policy) sharing convolution layers. It was reported that the joint training of the integrated network is effective to avoid the overfitting problem.

Note that a policy network for chess and shogi would be much more complex than that for Go — Although the average number of legal moves per position in Go is larger than that in chess or shogi, the total number of possible legal moves in all legal positions is much larger in chess or shogi than that in Go. The policy output is required to be able to present all the legal moves, which is the reason why the output layer of AlphaZero is so large in chess or shogi [2]. In order to avoid this problem, we designed a joint training scheme with a simple discrimination network instead of a policy network.

As well as the combined value and policy networks in AlphaGo Zero [4], other researchers also reported that multi-task learning should be beneficial for jointly-trained single tasks. L. Kaiser et al. [13] demonstrated that joint training largely helped tasks with less data learn from other tasks, while critical parts of their work included a multi-modal architecture in which as many parameters as possible are shared and the use of computational blocks from different domains together. Z. Zhang et al. [14] optimized facial landmark detection for a tasks-constrained deep model, together with heterogeneous but subtly correlated tasks. Their task-constrained learning succeeded in outperforming existing methods and reducing model complexity.

Typical standard regularization methods that have been widely applied may include sparsity-based L_1 regularization, weight decay-based L_2 regularization, and dropout techniques for neural networks. The L_2 regularization [3] improves generalization by choosing smaller network parameters to solve learning problems. With appropriate configurations, it usually performs better than other methods in practice, and thus has been adopted by many computer chess and shogi programs, including AlphaZero [2] and Bonanza [10]. Dropout [15] is also effective in the training of deep neural networks to prevent overfitting by randomly disabling neurons during training. However, larger networks require more extra training time.

As a representative of the standard methods, we conducted comparative experiments between the L_2 and uniformity regularization in this paper. The results suggest that the proposed method is more effective in specific applications.

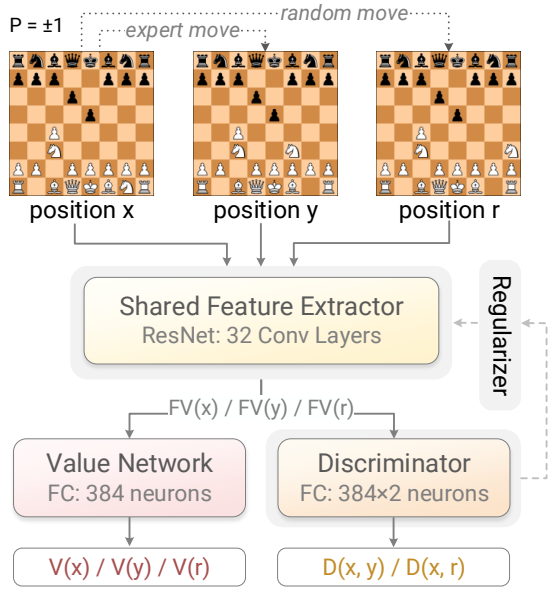


Fig. 1. Network Structure and Data Flow Chart

III. Training with Uniformity Regularizer

This section describes our training scheme including the structures and objective functions of our new regularization network and the value network we used in experiments. The network structure and main data flow are shown in Figure 1.

As described in the introduction, the main objective of this paper is to train a value network that evaluates the advantage of a given game position. We assumed that convolutional layers are effective in building evaluation functions for a target game, as in [4]. We also assume that game records are given as training data, as in an iteration in reinforcement learning [2] or supervised learning [10].

A. Network Structure

We propose to train a value network with a uniformity regularizer, which introduces two modifications to the standard training method: (1) to train jointly with a discrimination network, and (2) to minimize the uniformity regularizer's loss. The joint training with a discriminator network is designed to promote the training of the value network through the shared convolution layers. Both networks are trained simultaneously and share a common feature extractor so that feature representation ability is enhanced while the time consumption for training does not significantly increase. The uniformity regularizer's loss is built upon the discrimination network, which aims to reduce the overfitting problem for both networks.

The value network $V(s)$ outputs a scalar float score for the input position s, with no activation function by default. Larger scores suggest the first player is in a dominant position, while smaller scores suggest the second player is more likely to win in that position. We believe that this definition of a value network is general enough for a wide number of applications. We demonstrate how the proposed method consistently works with various kinds of loss functions for training of value networks in experiments.

The discriminator network $D(s, s')$ is trained to predict the certainty that a move played in two successive positions s and s' is played by an expert. So it takes two positions as its input. The output is assumed to be $[-1, 1]$, where 1 (-1) means most (least) certain. Unlike the value network that predicts winning probability quantitatively, the discriminator classifies moves qualitatively.

This discriminator is incorporated from the design of style transfer generative adversarial networks (STGANs) [16], where it is used to judge if the move between two successive positions is in accordance with specific top players' styles and to influence another independent value network through a common loss function. Although the discriminator itself has the same functionality in both our method and STGANs, but training goals and loss functions are completely different.

The uniformity regularization (UR) network is the jointly trained discriminator with a specially designed regularization function, which ensures all samples are discriminated as uniformly as possible, and enable the network to correctly evaluate a majority of positions rather than specializing on a few high-frequency but unimportant samples. Although the function is applied to the discriminator, it can also influence the value network through the shared convolutional layers.

B. Loss Functions for Uniformity Regularizer

When game records are available for training data, we can sample and pre-processe them to make every instance in the training dataset contains a quadruple $\langle x, y, r, p\rangle$, where x is a game position randomly selected from the game records and y, r are the subsequent positions after a recorded move (e.g., an expert move in supervised training) and a random but legal move respectively. p represents the player to move at the turn of position x. $p = 1$ and -1 for the first and the second player, respectively.

1) Overall: The value and the discriminator networks are trained simultaneously by using an integrated objective function $J_{\text{All}}(\theta)$:

$$J_{\text{All}}(\theta) = J_{\text{V}}(\theta) + J_{\text{D}}(\theta) + J_{\text{UR}}(\theta) \tag{1}$$

where $J_{\text{V}}(\theta)$, $J_{\text{D}}(\theta)$ and $J_{\text{UR}}(\theta)$ are the loss functions for the value network, the discriminator network, and the uniformity regularizer, respectively.

2) Discriminator: The discriminator loss is defined as the difference between the certainty of the move played in a real game record y and that of a random move r at position x:

$$J_{\text{D}}(\theta) = \frac{1}{n}\sum_{i=1}^{n}[D(\langle x_i, y_i\rangle) - D(\langle x_i, r_i\rangle)) - 2]^2 \tag{2}$$

where θ refers to the parameters of the joint neural networks, n is the batch size of training data, and $\langle x_i, y_i, r_i\rangle$ represents the i^{th} instance in that batch. The same notations are used in the following equations. The minimization of Eq. (2) encourages $D(\langle x_i, y_i\rangle)$ to be 1 and $D(\langle x_i, r_i\rangle))$ to be -1.

3) Uniformity Regularizer: The loss function of the uniformity regularizer is defined by the difference in the discriminator losses among samples:

$$J_{\mathrm{UR}}(\theta) = \frac{1}{m}\sum_{i=1}^{m}[(D(\langle x_i, y_i\rangle) - D(\langle x_{m+i}, y_{m+i}\rangle))^2 + (D(\langle x_i, r_i\rangle) - D(\langle x_{m+i}, r_{m+i}\rangle))^2] \quad (3)$$

where $m = n/2$ is the half of the batch size.

Since all instances in training batches have been randomly shuffled in advance, Eq. (3) is equivalent to measure and minimize the difference between the discriminated results of two randomly sampled positive (negative) samples, which is beneficial for reducing the variance of discriminated results, as well as maintaining the consistency of extracted features.

It should be noted that the value network is required to be as precise as possible to determine the best move from many legal moves and/or to predict the win/loss probability for a given position. Therefore, applying the same regularization on the value network does not provide any improvement and may even weaken its accuracy. Conversely, the discriminator only outputs qualitative judgments, not sensitive to specific numbers. Thus, it is more suitable for the proposed qualitative regularization.

4) Objective Functions of Value Networks and Implementation: Various objective functions and training methods have been proposed for the training of value networks. To examine the effectiveness and applicability of the proposed regularization, three objective functions are selected:

$$J_{\mathrm{Sigmoid}}(\theta) = \frac{1}{n}\sum_{i=1}^{n}\sigma(p_i(V(r_i) - V(y_i))) \quad (4)$$

$$J_{\mathrm{STGANs}}(\theta) = -\frac{1}{n}\sum_{i=1}^{n}[\ \log(\sigma(V(x_i) - V(y_i))) + \log(\sigma(V(y_i) - V(x_i))) + \log(\sigma(p_i(V(y_i) - V(r_i))))\ ] \quad (5)$$

$$J_{\mathrm{Squares}}(\theta) = \frac{1}{n}\sum_{i=1}^{n}[\ (V(x_i) - V(y_i))^2 + (V(y_i) - V(r_i) - 1)^2\ ] \quad (6)$$

where $\sigma(x) = 1/(1 + \exp(-x))$ is the sigmoid function, and p_i represents the player to move in the i^{th} instance.

Equation (4) is based on the objective function in the training of the evaluation function of the shogi program Bonanza [10], which won first place at the World Computer Shogi Championship in 2006 and 2013. This function encourages that the value of the position after an expert move $V(y_i)$ should be better than that after a random move $V(r_i)$. The similar loss function is used in comparison training in chess [9]. Equation (5) originates from the chess program Deep Pink [17] and was modified by the authors of STGANs [16]. This function encourages that the value of the position after an expert move $V(y_i)$ should be better than that after a random move $V(r_i)$, and be similar to that of the original position $V(x_i)$. Equation (6) was adopted in our previous study [18] inspired by least squares generative adversarial networks (LSGANs) [19]. This function shares the same purpose as in Eq. (5) but represents it in a least squares manner.

Moreover, we believe that our uniformity regularizer will successfully work with other loss functions including TD(λ) as in [6] and Monte-Carlo returns (win/loss results) as in [2]. We chose the above three functions here so as to discuss the effectiveness with respect to a limited amount of game records.

C. Data Flow and Representation

For the implementation of the input and convolution layers, we basically followed the design of AlphaZero [2]. Initially, a chess position is represented as a 13-channel image of 8×8 pixels, corresponding to 12 piece types and the player to move. Similarly, a shogi position is composed of 43 channels of 9×9 pixels, corresponding to 42 piece types and the current player, including 14 channels for prisoner pieces.

The feature extractor is a residual convolutional network [20] composed of 16 standard residual blocks. There are 128, 192, 256 and 384 filters in the $1st$, $2nd$, $3rd$ and $4th$ four blocks successively. It accepts a game position as input and outputs a 384-dimensional flatten vector as extracted high-level features (i.e. FV$(s), s \in [x, y, r]$ in Figure 1). The features are then input into the fully connected layers of the value network and the discriminator network respectively. All hidden layers are activated by rectified linear unit functions [21].

For the discrimination network, a 768-dimensional vector concatenated by two feature vectors yielded by the convolution layers for two positions is given as the input into its last fully connected layer, and output as a float number constrained in $[-1, 1]$ by a hyperbolic function $tanh(x)$.

Following the previous work in STGANs [16], we do not constrain the range of output values for our value networks.

IV. Experiments

To demonstrate how the proposed method contributes to yielding accurate value networks for evaluation functions, comparative experiments were conducted, including the robustness of our method with respect to objective functions, comparison with the L_2 regularization, and performances tests on different sizes of training data. The main experiments were conducted with chess, but results for shogi are also shown.

A. Dataset and Configurations

We prepared a dataset containing 783,129 games downloaded from the computer chess database CCRL 40/40 (http://www.computerchess.org.uk/ccrl/4040, acquired on January 14, 2018) and 868,161 games from the computer shogi server Floodgate (http://wdoor.c.u-tokyo.ac.jp/shogi/index-e.html, acquired on December 31, 2017). 85% of the data were used for training, and the remaining 15% were for testing. Only moves made by the winner were adopted in the training and test datasets. All positions in the datasets were randomly shuffled in advance. Although a considerable amount of data was acquired, it is much less than that for AlphaZero [2], in which 44 million training games for chess

TABLE I
PREDICTION ACCURACY OF TRAINED NETWORKS FOR CHESS

Model	*1-1 Acc.*	*Top-1 Acc.*	*Speed**
Sigmoid (V+D+UR)	91.78%	40.32%	16.60
Sigmoid (V+D)	91.40%	39.25%	16.70
Sigmoid (V)**	–	–	18.76
Squares (V+D+UR)	91.09%	38.27%	16.67
Squares (V+D)	90.92%	37.72%	16.63
Squares (V)	90.10%	35.87%	16.71
STGANs (V+D+UR)	90.69%	37.26%	16.36
STGANs (V+D)	90.39%	36.36%	16.43
STGANs (V)	88.95%	33.49%	16.90

* Unit: Training positions per second.

** The model collapsed due to gradient exploding.

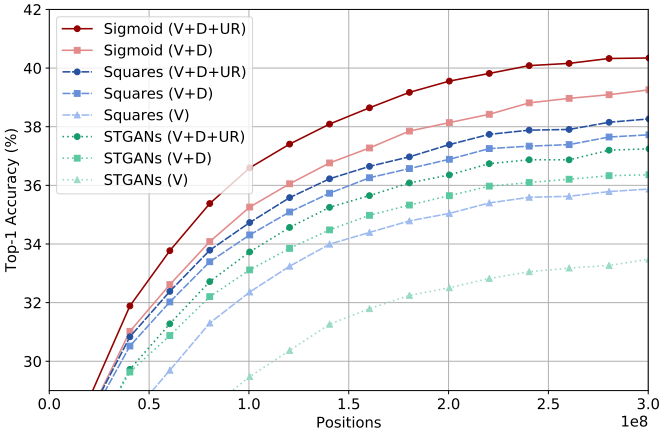


Fig. 2. Top-1 Accuracy of Trained Networks in Chess

TABLE II
CHANGES AFTER APPLYING REGULARIZATION METHODS

Regularizer	*1-1 Acc.*	*Top-1 Acc.*	*Norm*
No Regularizer *	91.40%	39.25%	88.56
Uniformity Reg'r **	+0.38%	+1.07%	+0.01
L_2 ($\lambda = 10^{-7}$)	+0.11%	+0.06%	−0.01
L_2 ($\lambda = 10^{-6}$)	+0.10%	+0.26%	+0.01
L_2 ($\lambda = 10^{-5}$)	+0.05%	+0.22%	−0.01
L_2 ($\lambda = 10^{-4}$)	+0.09%	+0.25%	+0.02
L_2 ($\lambda = 10^{-3}$)	+0.15%	+0.40%	+0.05
L_2 ($\lambda = 10^{-2}$)	+0.07%	+0.18%	+0.05
L_2 ($\lambda = 10^{-1}$)	+0.06%	+0.17%	+0.02
L_2 ($\lambda = 10^{0}$)	+0.08%	+0.26%	+0.03
L_2 ($\lambda = 10^{1}$)	+0.02%	+0.14%	−0.01
L_2 ($\lambda = 10^{2}$)	+0.15%	+0.38%	+0.02
L_2 ($\lambda = 10^{3}$)	+0.08%	+0.12%	+0.00

* Sigmoid (V+D), the baseline model.

** Sigmoid (V+D+UR), the proposed model.

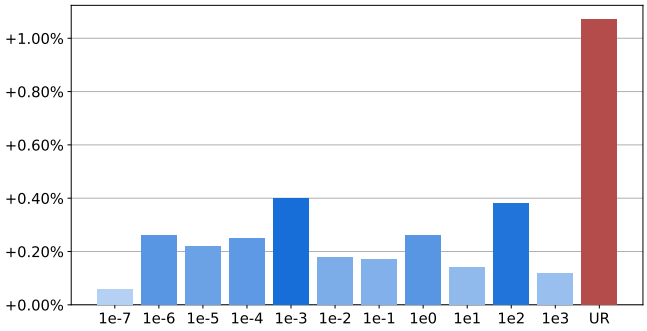


Fig. 3. Top-1 Accuracy Improved by L_2 and Uniformity Regularization

and 24 million games for shogi were involved. In this context, applying appropriate regularization techniques is of significant importance.

By default, models were trained in 4 processes on a single machine with two NVIDIA 1080Ti GPUs. A stochastic gradient descent optimizer with a momentum rate of 0.9 were used. The learning rate starts from 0.001 and decays every 10 000 steps with a base of 0.99. The batch size n was set as 128. The weights in neural networks were randomly initialized by normal distribution with mean zero and variance one.

B. Robustness in Objective Functions of Value Networks

To demonstrate that the proposed method could cooperatively work with various loss functions, three models `Model` ∈ [`STGANs`, `Sigmoid`, `Squares`] introduced in section III-B4 were trained, with the following conditions:

- `Model(V)` is a plain value network with the objective J_{Model}, adopted as a baseline model in experiments. Neither a discriminator nor regularization is applied.
- `Model(V+D)` consists of the above value network and a discriminator (i.e. to minimize $J_{\text{All}}(\theta) = J_{\text{V}}(\theta) + J_{\text{D}}(\theta)$). No regularization is applied.
- `Model(V+D+UR)` consists of the above networks and trained with the uniformity regularization (i.e., $J_{\text{All}}(\theta) = J_{\text{V}}(\theta) + J_{\text{D}}(\theta) + J_{\text{UR}(\theta)}$ is minimized).

For evaluating the performance of a model, the following two metrics are used in this paper.

- **1-1 accuracy**: For two given positions y, r after an expert move and a random move respectively, the output of a value network is correct if $V(y) > V(r)$ when $p = 1$ or $V(y) < V(r)$ when $p = -1$.
- **Top-k accuracy**: For a given position x and its subsequent position y after an expert move, the output of a value network is correct if $V(y)$ is one of the k highest scores among all subsequent positions of x when $p = 1$, or one of the k lowest scores when $p = -1$.

We trained 9 value networks with 300 million positions that were randomly sampled from over 700,000 games in total, and show their accuracies on the test dataset in Table I and Figure 2. The test accuracies shown in figures are averages in the last 300 minutes during the training process.

It can be found that both the proposed joint training and the uniformity regularizer are effective and compatible with all the 3 objective functions. On average, the top-1 accuracy of a

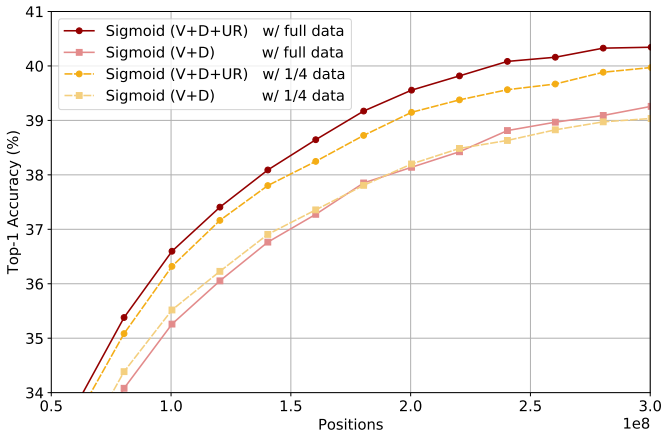


Fig. 4. Top-1 Accuracy on Complete and Reduced Datasets

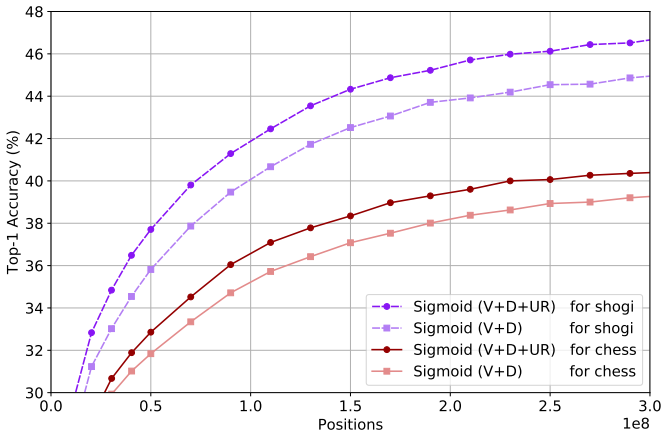


Fig. 5. Top-1 Accuracy in Chess and Shogi

plain model is improved by 2.36% after the joint training, and a further 0.84% after the uniformity regularization is applied.

The data amount and update frequency at each step is the same for all configurations, thus the accuracy improvement between `Model(V)` and `Model(V+D)` mainly comes from structural changes (i.e., joint training with shared convolution layers). It is consistent with previous work in [8] and [10] that models with Eq. (4) performed better than the others. The `Sigmoid` objective function in Eq. (4) should be beneficial for reducing vanishing gradient problems in DCNNs, and correspondingly, it may also cause exploding gradient problems when high learning rates are applied. `Sigmoid(V)` collapses with the default network structure and learning rate, while the introduction of the discriminator in `Sigmoid(V+D)` successfully prevents any severe exploding gradient problem in the model.

According to the results, the joint training of multiple tasks should be considered as capable of enhancing the representation ability of the feature extractor, having a significant regularization effect on it. The models with the uniformity regularization performed better than those with ordinary joint training.

The training speed of each model is listed in the last column of Table I. For the objective function $J_{\mathrm{V}}(\theta)$ whose complexity is equal to or greater than $J_{\mathrm{D}}(\theta)$ and $J_{\mathrm{UR}}(\theta)$, the joint training and the uniformity regularization do not increase time costs significantly.

C. Comparison with L_2 Regularization

The uniformity regularization served as a substantial enhancement to the joint training. We further compared the effectiveness of the uniformity regularization with that of the L_2 regularization, which is a simple and effective method, with a wide range of applications.

Formally, a cost function with a L_2 regularization term is defined as:

$$J(\omega) = J_0(\omega) + \frac{1}{2}\lambda \sum_i \omega_i^2 \tag{7}$$

where ω refers to the weight vector of network parameters, $J_0(\omega)$ is the original objective, the added last term is the norm of θ, and λ is a parameter determining how strongly large weights are penalized.

Although the best model does not always have the minimum norm of its weight vector, it is a common practice to incorporate a L_1 and/or a L_2 regularizer(s) in the training of large networks as [2] and in comparison training as [10]. Therefore, we investigated here whether our jointly trained network can work consistently with the standard L_2 regularization, and how different the effects of the L_2 and the uniformity regularizers are. We do not argue if the L_2 regularization is appropriate for our loss function or training method in all circumstances, as the answer depends on actual data and network structures.

To determine the best configuration for the L_2 regularizer, 11 models were tested in total with λ values ranging from 10^{-7} to 10^{3}. Since $J_{\mathrm{Sigmoid}}(\theta)$ achieves the highest accuracy in previous tests, `Sigmoid (V+D)` was adopted as the baseline model for comparing regularizers. The changes of test accuracy and global norms for all parameters are listed in Table II and Figure 3.

After 300 million positions were trained, the L_2 regularization obtained its highest top-1 accuracy of 39.65% at $\lambda = 10^{-3}$. However, there is still a significant difference between the L_2 and the uniformity regularization methods.

The goal of the L_2 regularization is to improve the training results by lowering the norm of weight vectors, while a lower norm does not always result in a better performance. We observed that the total dimension of all parameters in our network is more than 19 million, and the average weight of each dimension was already very close to 0 after training, which suggests there was no enough space to improve performance by directly constraining the parameter norm. To the contrary, the uniformity regularizer mainly emphasizes the correctness of the qualitative relationship among discriminated results, instead of adjusting weights blindly or brutally. Therefore, both regularization and acceleration can be achieved simultaneously.

We observed that the accuracy difference in the training

TABLE III
PUBLISHED PREDICTION ACCURACY OF RELATED WORK

Accuracy	*Proposed* Chess	*Giraffe* Chess	*Proposed* Shogi	*Bonanza* Shogi
1-1	92.12%	–	97.34%	–
Top-1	41.54%	45.73%	47.81%	37%
Top-2	60.96%	61.68%	66.93%	–
Top-3	72.14%	69.59%	76.83%	–
Top-4	79.38%	74.77%	82.85%	–
Top-5	84.34%	78.48%	86.81%	–
Top-6	87.93%	83.86%	89.58%	–
Top-7	90.57%	85.63%	91.58%	–
Top-8	92.59%	87.49%	93.08%	–
Top-9	94.13%	88.84%	94.21%	–
Top-10	95.53%	90.38%	95.11%	–

TABLE IV
PREDICTION ACCURACY FOR THE SAME CHESS DATASET

Accuracy	*Proposed* Chess	*Giraffe* Chess	*Stockfish* Chess
1-1	91.60%	–	–
Top-1	39.28%	32.89%	34.24%
Top-2	58.69%	49.51%	51.47%
Top-3	70.03%	59.61%	62.21%
Top-4	77.48%	66.78%	69.59%
Top-5	82.69%	72.06%	75.14%
Top-6	86.52%	76.26%	79.41%
Top-7	89.37%	79.63%	82.77%
Top-8	91.61%	82.52%	85.54%
Top-9	93.34%	85.03%	87.89%
Top-10	94.71%	87.19%	89.82%

and test data were usually within 0.4% for all configurations of `Sigmoid (V+D)` and `Sigmoid (V+D+UR)`. Therefore, the performance improved by the uniformity regularizer can be explained not only by the prevention of overfitting to a part of training samples but also by a better understanding to covered relationships among them.

D. Performance on Reduced Dataset

To understand the properties of the proposed method and validate its performance on different data scales, a quarter of the training games were randomly selected from the complete chess dataset. We then trained `Sigmoid (V+D)` and `Sigmoid (V+D+UR)` and observed their training curves by using the smaller dataset while keeping the test dataset unchanged.

Test results are shown in Figure 4. Within 300 million training positions, `Sigmoid (V+D+UR)` with a quarter of data performed better than `Sigmoid (V+D)` and achieved a relatively high top-1 accuracy of 39.97%. However, the difference between `Sigmoid (V+D)` and `Sigmoid (V+D+UR)` decreases when the training dataset shrinks. The major cause of this phenomenon is the diversity of training data. The networks are fed the same number of positions in both configurations. However, when the total number of game records decreases, the average number of positions sampled from the same game increases. Since the uniformity regularization function $J_{\mathrm{UR}}(\theta)$ relies on the instances existing in the training dataset, the generalization ability of the uniformity regularizer could be affected if available independent patterns in dataset are reduced.

Existing standard regularization methods including the L_2 regularization are mainly applied by constraining the weights of the network parameter directly, while the uniformity regularization influences the parameter by using training data as a medium. Due to this property, the performance of the proposed method may be restricted when samples are limited, while it is also promising to bring substantial improvement to value networks when data are relatively sufficient.

E. Application to Shogi

To examine the application scope of the proposed method, value networks for shogi were trained in the same manner as the models for chess.

Experimental results are plotted in Figure 5. At the time that 300 million positions were trained, the accuracy of the uniformity regularization over the baseline improved by 1.70% for shogi, which is even larger than that by 1.09% for chess. This is attributed to two reasons:

- Value networks in shogi require more complex feature maps than those in chess, being more prone to overfitting and requiring the application of appropriate regularization methods.
- Shogi has larger state and decision spaces, as the amount of training instances generated per game is about 1.69 times more than chess, which is beneficial for alleviating data sparsity and boosting the effects of the uniformity regularization.

F. Comparison with Related Work

To acquire more stable performance data, we measured the proposed value networks `Sigmoid (V+D+UR)` in chess and shogi, respectively, after 600 million training positions (twice that of the above experiments) were trained.

Table III shows the prediction accuracy of our network, and those of Giraffe [7] and Bonanza [10] that are referenced from their official publications. It should be noted that these networks were trained with different positions and the accuracies were also measured separately upon irrelevant datasets. Two kinds of neural networks were trained for Giraffe: one for its evaluation function and the other works as a policy network. We wanted to compare the evaluation network of Giraffe with our value network, however, only the data of its policy network were available. Nonetheless, these data are still important references, which basically reflect the best performance that the networks can reach on their preferred test datasets. The top-1 accuracy of the proposed shogi value network largely surpassed the linear model with quiescence search of Bonanza by about 11%. Despite the top-1 and top-2

accuracies of the proposed chess value network were lower than those of Giraffe, all its data below the top-3 accuracy were higher than Giraffe.

In addition to comparing the published performance data, we also evaluated available models with our dataset. We collected extra $20,740$ game records from CCRL 40/40, released within the period from January 15 to March 15, 2018. $4,163$ games played by top-level programs with ratings higher than $3,200$ were selected to form the new test dataset. Only moves made by the winner were included in the dataset.

We tested the performances with the new dataset for the proposed chess model, Giraffe with network parameters officially released on September 8, 2015 [22], and the most advanced open source chess program Stockfish 9 [23]. We only focused on the evaluation scores generated by Giraffe and Stockfish, while the influence of search algorithms were intentionally excluded from the experiments. The measured accuracies are shown in Table IV. The proposed model achieved the highest prediction accuracies on the uniform dataset against the others.

Note that this does not indicate that our value network is better than Stockfish in terms of playing strength, as the execution speed of our network is much slower than the ordinary methods of other neural networks. Therefore, it should be noted that Stockfish places more importance on search efficiency. We also conducted an experimental game between our network and Stockfish (without search). The game started from the initial position and ended in a draw. For acquiring thorough assessment results, one of our future work is to test the networks through the games with fair and various opening positions from published databases.

Generally, although a completely fair comparison is difficult, the above experimental results are sufficient to prove that our value networks trained with the uniformity regularizer reached a meaningful level in move prediction for chess and shogi.

V. Conclusion

In this paper, we presented a novel uniformity regularization network, as well as a newly designed joint training scheme for building high-accuracy evaluation functions for chess and shogi. Through regularization applied to the shared convolutional feature extractor, the prediction accuracy of the value network in evaluation functions can be significantly improved. Experimental results demonstrated that the proposed regularization is capable to outperform the standard L_2 regularization method, and enable the value network to reach a decent level in predicting moves.

The applications of DCNNs and multi-task joint training have yet to be widely introduced into the research fields of chess variants and other board games. We believe the training scheme and experimental data can be provided as a useful reference for follow-up studies.

Acknowledgement

A part of this work was supported by JSPS KAKENHI Grant Number 16H02927 and by JST, PRESTO.

References

[1] M. Campbell, A. J. Hoane Jr, and F.-h. Hsu, "Deep blue," *Artificial intelligence*, vol. 134, no. 1-2, pp. 57–83, 2002.

[2] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel *et al.*, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," *arXiv preprint arXiv:1712.01815*, 2017.

[3] A. Krogh and J. A. Hertz, "A simple weight decay can improve generalization," in *Advances in neural information processing systems*, 1992, pp. 950–957.

[4] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, p. 354, 2017.

[5] M. Buro, "From simple features to sophisticated evaluation functions," in *International Conference on Computers and Games*. Springer, 1998, pp. 126–145.

[6] J. Baxter, A. Tridgell, and L. Weaver, "Learning to play chess using temporal-differences," *Machine Learning*, vol. 40, no. 3, pp. 242–263, Sep. 2000.

[7] M. Lai, "Giraffe: Using deep reinforcement learning to play chess," *arXiv preprint arXiv:1509.01549*, 2015.

[8] G. Tesauro, "Connectionist learning of expert preferences by comparison training," in *Advances in neural information processing systems*, 1989, pp. 99–106.

[9] ——, "Comparison training of chess evaluation functions," in *Machines that learn to play games*. Nova Science Publishers, Inc., 2001, pp. 117–130.

[10] K. Hoki and T. Kaneko, "Large-scale optimization for evaluation functions with minimax search," *Journal of Artificial Intelligence Research*, vol. 49, pp. 527–568, 2014.

[11] O. E. David, N. S. Netanyahu, and L. Wolf, "Deepchess: End-to-end deep neural network for automatic learning in chess," in *International Conference on Artificial Neural Networks*. Springer, 2016, pp. 88–96.

[12] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[13] L. Kaiser, A. N. Gomez, N. Shazeer, A. Vaswani, N. Parmar, L. Jones, and J. Uszkoreit, "One model to learn them all," *arXiv preprint arXiv:1706.05137*, 2017.

[14] Z. Zhang, P. Luo, C. C. Loy, and X. Tang, "Facial landmark detection by deep multi-task learning," in *European Conference on Computer Vision*. Springer, 2014, pp. 94–108.

[15] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[16] M. Chidambaram and Y. Qi, "Style transfer generative adversarial networks: Learning to play chess differently," *arXiv preprint arXiv:1702.06762*, 2017.

[17] E. Bernhardsson. (2014) Deep learning for chess. [Online]. Available: https://erikbern.com/2014/11/29/deep-learning-for-chess.html

[18] S. Wan and T. Kaneko, "Imitation learning for playing shogi based on generative adversarial networks," in *Conference on Technologies and Applications of Artificial Intelligence*, 2017.

[19] X. Mao, Q. Li, H. Xie, R. Y. Lau, Z. Wang, and S. P. Smolley, "Least squares generative adversarial networks," in *2017 IEEE International Conference on Computer Vision (ICCV)*. IEEE, 2017, pp. 2813–2821.

[20] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[21] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, 2011, pp. 315–323.

[22] L. Matthew. (2016) Giraffe: An experimental chess engine based on deep learning. [Online]. Available: https://bitbucket.org/waterreaction/giraffe/wiki/Home

[23] T. Romstad, M. Costalba, and J. Kiiski. (2018) Stockfish: A strong open source chess engine. [Online]. Available: https://stockfishchess.org/

Regular Language Inference for Learning Rules of Simplified Boardgames

Jakub Kowalski
Institute of Computer Science, *University of Wrocław*
Wrocław, Poland
jko@cs.uni.wroc.pl

Andrzej Kisielewicz
Institute of Mathematics, *University of Wrocław*
Wrocław, Poland
Andrzej.Kisielewicz@math.uni.wroc.pl

***Abstract*—We deal with the problem of learning game rules by observing the play, the study initiated by Björnsson for the class of Simplified Boardgames, describing a rich family of chess-like games. In this paper we restate the problem in terms of regular language inference and improve Björnsson's algorithm combining applications of existing DFA learning algorithms with our domain-specific approach. We present the results of tests on a number of games, including human-made and artificially generated ones.**

***Index Terms*—general game playing, regular language inference, simplified boardgames, deterministic finite automata**

I. Introduction

The aim of *General Game Playing* (GGP) [1] is to develop a system that can play a variety of games with previously unknown rules. Unlike standard AI game playing, where designing an agent requires special knowledge about the game, in GGP the key is to create a universal algorithm performing well in various situations and environments. After the launch of the annual International General Game Playing Competition (IGGPC) in 2005 [2], [3], many new languages have been developed to describe certain classes of games [4], [5], [6], [7] and other competitions have been proposed [8]. Identified as a new Grand Challenge of Artificial Intelligence, GGP consists of many research challenges requiring combining a number of domains, such as knowledge representation, searching, planning, reasoning, and machine learning [9].

In [4], Björnsson has proposed a new scenario, where the game rules should be learned by observing others play, rather than obtained from a given description. This partially coincides with the rules of GVG-AI competition [8], where a prepared reasoner (java object allowing state manipulation) is given instead of the raw game rules. Björnsson's approach concentrates on learning deterministic finite automata (DFA) which are used to encode learned rules in a class of games introduced by the author and called Simplified Boardgames. The proposed class is substantially narrower than GGP systems mentioned before, yet more general and concise than previous such approaches [10], [11]. As the DFA representation favors the efficiency of game state manipulation, possibilities of implementing such mechanism to support GGP players have been considered.

In this paper, we deal with the problem of learning game rules by observing. We continue the study initiated by Björnsson in [4] for the class of Simplified Boardgames, focusing on efficient learning of the observed boardgame moves. First, we consider applications of existing DFA learning algorithms for the task of learning piece movements from the set of game records and restate the problem in terms of Regular Language Inference [12]. Secondly, we propose our domain-specific algorithms to ensure better efficiency and a higher chance of obtaining a correct approximation of the actual DFA, assuming incompleteness of given training data. We test all presented approaches combining various learning algorithms with different procedures checking DFA consistency in a number of games, both human-made and artificially generated. Following Björnsson's approach we consider two types of training data: GGP-like, where for every position all legal moves are listed, and human-play-like, providing more sparse data where only the move actually made by a player has been recorded.

The paper is organized as follows. Section II provides the necessary background for the class of Simplified Boardgames, learning by observing, and Regular Language Inference. In Section III, we formally state the problem and analyze selected existing approaches from the theoretical point of view. In Section IV, we introduce new consistency checking procedures, and in Section V we present our new algorithm learning piece's moves by observing. Finally, Section VI presents the results of experiments evaluating the performance of all the considered approaches.

II. Preliminaries

In this section we introduce domains relevant to our study, providing necessary algorithms and terminology.

Let Σ^* be the set of all possible words over the alphabet Σ. For DFA $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F \rangle$, where Q is the set of states, q_0 is the initial state, $F \subseteq Q$ is the set of accepting states, and $\delta : Q \times \Sigma \to Q$ the transition function (which, in our study, is usually a partial function), by $L(\mathcal{A})$ we denote the language accepted by $\mathcal{A}$.

A. Simplified Boardgames

Simplified Boardgames is the class of fairy chess-like games introduced by Björnsson in [4]. The language describes turn-based, two-player, zero-sum chess-like games on a rectangular board with piece movements described by regular languages and independent on the move history. It was slightly extended

978-1-5386-4359-4/18/$31.00 ©2018 IEEE

in [13], and used as a comparison class for assessing the level of Stanford's GGP programs.

Here we follow the class formalization from [14] to provide a shortened necessary introduction. The game is played between the two players, *black* and *white*, on a rectangular board of size *width*×*height*. White player is always the first to move. Although it may be asymmetric, the initial position is given from the perspective of the white player, i.e. forward means "up" for white, and "down" for black.

During a single turn, the player has to make a move using one of his pieces. Making a move is done by choosing the piece and changing its position according to the specified movement rule for this piece. At any time, at most one piece can occupy a square, so finishing the move on a square containing a piece (regardless of the owner) results in removing it (capturing). No piece addition is possible. After performing a move, the player gives control to the opponent.

For a given piece, the set of its legal moves is defined as the set of words described by a regular expression over an alphabet Σ containing triplets $(\Delta x, \Delta y, on)$, where Δx and Δy are relative column/row distances, and $on \in \{e, p, w\}$ describes the content of the destination square: e indicates an empty square, p a square occupied by an opponent piece, and w a square occupied by an own piece. A positive Δy means forward, which is a subjective direction and differs in meaning depending on the player.

Consider a piece and a word $w \in \Sigma^*$ that belongs to the language described by the regular expression in the movement rule for this piece. Let $w = a_1 a_2 \dots a_k$, where each $a_i = (\Delta x_i, \Delta y_i, on_i)$, and suppose that the piece stands on a square $\langle x, y \rangle$. Then, w describes a move of the piece, which is applicable in the current board position if and only if, for every i such that $1 \leq i \leq k$, the content condition on_i is fulfilled by the content of the square $\langle x + \sum_{j=1}^{i} \Delta x_j, y + \sum_{j=1}^{i} \Delta y_j \rangle$. The move of w changes the position of the piece piece from $\langle x, y \rangle$ to $\langle x + \sum_{i=1}^{k} \Delta x_i, y + \sum_{i=1}^{k} \Delta y_i \rangle$. An example of how move rules work is shown in Figure 1.

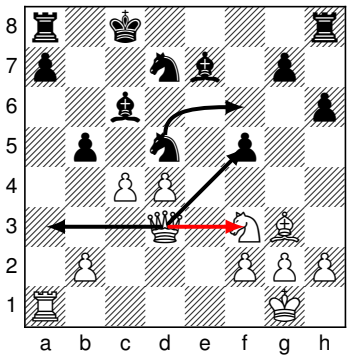


Fig. 1. A chess example. Two legal moves for the queen on $d3$ are shown. The capture to $f5$ is codified by the word $(1, 1, e)(1, 1, p)$, while move to $a3$ is encoded by $(-1, 0, e)(-1, 0, e)(-1, 0, e)$. The move to $f3$ is illegal, as in the language of queen's moves no move can end on a square containing own's piece. The $d5 - f6$ knight move is a direct jump codified by the one-letter word $(2, 1, e)$.

B. Learning by Observing

The initial purpose of introducing Simplified Boardgames was to study capabilities of learning game rules, given records of previously played games [4]. As terminal conditions were deliberately kept simple, the proper task comes to learning the languages of piece movements. This extends the directions of the GGP research, beforehand focused mainly on learning how to play a given game.

Two types of training data were proposed: a real-world *single move known* scenario – where the observer sees only the performed move, and a GGP-like *all legal moves known* scenario – where for every position all possible legal moves are visible. The latter case reflects the situation when we want to learn rules of the game given only the reasoner able to compute moves, advance game position, and detect the terminal states. Such novel, entirely simulation-based approach to general playing is used for General Video Game Playing Competition.

In [4], the algorithm for learning a DFA consistent with the observed game positions has been presented. To improve learning rate, some additional assumptions were made, allowing unseen moves to be accepted as correct if certain conditions were met. An extended approach, using slightly modified Simplified Boardgames domain, has been recently presented in [15]. The language has been modified to support basic piece addition and deletion. The process of learning was performed using the LOCM acquisition system [16], [17], which is an inductive reasoning system that learns planning domain models from action traces.

Other approaches described in the literature are mainly logic-based. In [18], chess variant rules described as first-order logic programs are learned using positive and negative examples, background knowledge, and the theory revision. The system that learns board-based games through the video demonstration using logic and descriptive complexity is presented in [19]. Alternatively, an interaction based approach to learn rules of simple boardgames from the dialogue with a human user is described in [20].

C. Regular Language Inference

Regular Language Inference is the problem of finite automata identification using labeled samples: given a disjoint sets of words S_+ containing words belonging to the target language L, and S_- containing words that do not belong to L, we ask what the size of the minimal DFA consistent with these sets is [12]. Gold ([21]) has proven that this problem is NP-hard, and Pitt and Warmuth ([22]) showed that given labeled samples corresponding to automata with n states, there is no polynomial algorithm that guarantees output DFA with at most $n^{(1-\epsilon)\log\log(n)}$ states for any $\epsilon > 0$.

The problem may be efficiently solved by polynomial algorithms if sets S_+, S_- fulfill additional restrictions, i.e. they are somehow representative. The first algorithm, known as TB/Gold was introduced by Trakhtenbrot and Barzdin in 1973 [23], and rediscovered by Gold five years later [21]. In 1992, RPNI (Regular Positive and Negative Inference) algorithm was

proposed independently by Oncina and Garcia [24], and Lang [25]. It guarantees that the obtained DFA is consistent, and is equivalent to the target DFA if S_+, S_- contains the so-called *characteristic set*. Both theoretical analysis and experiments support the thesis that the results obtained by RPNI are better, and it converges faster than TB/Gold algorithm [26]. The pseudocode and detailed description of RPNI algorithm are provided in Section III-C. Several modifications of RPNI have been studied, e.g. incremental version [27], with faster convergence for some languages subfamilies [28], and suitable for PAC learning [29].

III. Boardgame Rules Inference

We are dealing with the situation where an agent observes a number of plays between some players and, by observing, it should learn the rules of this game. Assuming we restrict possible games to the class of Simplified Boardgames, where the main challenge consists of discovering the rules of piece movements, the problem can be stated as a specific variant of the Regular Language Inference.

On the one hand, this gives us a possibility to take advantage of well-known generally applicable algorithms. On the other hand, as the learning domain is narrow and characterized by certain properties, it should be possible to create more effective domain-specific learning algorithms.

A. Problem Statement and Model

Let Σ^* be the set of moves for a given Simplified Boardgame, i.e. it contains words over the alphabet of $(\Delta x, \Delta y, on)$ triplets. Given data obtained by observing movements of a piece p, there exist the partition of Σ^* into the following languages: the language S_+ of observed legal moves, the language S_- of known illegal moves, the language S_0 containing all words that are impossible to perform from any square on the board, and $S_?$ containing all the other words. So, if $w \in S_?$, then it is theoretically possible that w belongs to L_p (the language of legal movements of p), yet there is no evidence in the data that it is legal or not.

Let us consider learning scenarios proposed in [4]. In the *single move known* scenario the S_- set is always empty, and $S_?$ may be non-empty. In the *all legal moves known* scenario S_- is not empty, yet we can still have words in $S_?$.

To meet the reality of Simplified Boardgames, the aforementioned languages have to satisfy the following properties. Languages S_+, S_- and $S_?$ are finite and closed on taking substrings, i.e. for any $w \in S_+ \cup S_- \cup S_?$ every substring of w also belongs to $S_+ \cup S_- \cup S_?$. Language S_0 is infinite, and such that for all $w \in S_0$ and $a \in \Sigma$, we have $aw \in S_0 \wedge wa \in S_0$. Also, there exist a procedure $\mathcal{O} : \Sigma^* \to \{T, F\}$ which, for given w, decide in time $\Theta(|w|)$ if $w \in S_0$. (We can iterate through the word summing relative distances and checking if the board size was exceeded.)

To approximate the size of all substring-closed sets we present the following observation. We use standard $|S|$ notation to denote the cardinality of the set S, and $||S||$ to denote the sum of the lengths of words in S.

Observation 1. *Given board of size $n \times n$, and $S = S_+ \cup S_- \cup S_?$. We have that:*

$$2\sum_{k=1}^{n^2-1} 3^k \frac{(n^2-1)!}{(n^2-1-k)!} \leq |S| \leq n^2 \sum_{k=1}^{n^2-1} 3^k \frac{(n^2-1)!}{(n^2-1-k)!},$$

which estimates the total number of moves that can be made on such a board.

Notice that in the case of standard chess-like games, the number of legal moves (which is a superset of S_+) is very small compared to the left-hand side of (1). This causes that explicit occurrence search in S_- or $S_?$ is highly inefficient in comparison to checking set membership via $\mathcal{O}$ function.

For a given piece p, let $\mathcal{A}$ be a DFA approximating L_p based on given observations. It is required that $S_+ \subseteq L(\mathcal{A})$ and $L(\mathcal{A}) \cap S_- = \emptyset$. However, there is some freedom concerning relations between $L(\mathcal{A})$ and the remaining sets. Whether $L(\mathcal{A}) \cap S_0$ will be empty or not, in practice do not influence the results generated by $\mathcal{A}$. During the move generation phase, movements that are impossible to be made on the current board position are simply excluded.

The question whether $L(\mathcal{A})$ could contain some words (and which one) from $S_?$ depends on the chosen policy. It is safe to assume that every unobserved move is treated as illegal. This ensures that the player based on $\mathcal{A}$ will never produce a move that could cause e.g. an instant loss. On the other hand, admitting some words from $S_?$ allows to simplify the language definition and minimize produced DFA. Moreover, without extending the set of accepted words, it is impossible to obtain the optimal automaton given data containing only partial knowledge about L_p.

B. Björnsson's Algorithm

First, we make some observations concerning the algorithm for learning piece rules proposed by Björnsson in [4]. Because of lack of the space, we have to refer the reader to [4] for details. We describe only a general structure of this algorithm.

Two main parts of the algorithm are the automaton learning and the consistency checking. The automaton learning ([4, Algorithm 2]) begins with constructing for a given piece a *Prefix Tree Acceptor* (PTA) from the training data. The whole procedure is based on the priority queue, containing candidates for the smallest consistent DFA. The maximal number of iterations of the main while loop is limited by the *MaxExpansions* constant. In each step, the smallest DFA is taken from the queue, and it is compared against the smallest DFA obtained so far. We iterate through the pairs of states, which are then merged (collapsed) forming a new automaton. There is a constant K that prevents considering the states which are too distant from each other.

The resulting automaton is the subject of the consistency checking procedure ([4, Algorithm 1])[1]. An accepted definition

[1]In [4], the algorithm immediately returns *True* if the first of the considered positions fulfills the desired property. The line 8 of the algorithm containing "**return** $movesDFA \subseteq pos.moves(sq)$" should be rewritten to "**if** $(movesDFA \supset pos.moves(sq))$ **then return** $False$ **end if**".

of *consistency* with the training data is that DFA should generate all moves known to be legal, and no moves known to be illegal. It is checked straightforwardly in the all legal moves known scenario. To handle the single move known scenario additional assumptions have been made. An unobserved move is accepted if it was observed for some other piece or certain essential parts of this move were observed.

If the automaton passes the consistency test it is inserted into the priority queue. Since the merging procedure can return a non-deterministic automaton, an additional determinization procedure is used. The theoretical time complexity of the algorithm is exponential. More precisely, we have

Observation 2. *Let S_+ be the language accepted by the piece's prefix tree acceptor pt, and $\mathcal{C}(k, td)$ the complexity of consistency checking the given training data td and a DFA with k states. Then, assuming that the $MaxExpansions$ and K parameters are constant, the complexity of LearnDFA(pt,td) may be bounded by*

$$O(||S_+||(2^{||S_+||} + \mathcal{C}(||S_+||, td))).$$

The part $2^{||S_+||}$ is the worst case rarely achieved in practice. The dominant part is the consistency checking, which requires browsing through all the observed positions, generating movements, and performing subset checking operations.

An important observation is that in some situations, DFA's returned by [4, Algorithm 2] generate illegal moves. For example, consider the *limited rider* piece that moves in one direction for a given limited distance. Examples of such pieces, like Short Rook or Cloud Eagle can be found in various fairy-chess games [30]. Let $\mathcal{D}$ be a training data in the *all legal moves* scenario, such that for a limited rider $\mathcal{Q}$ all its legal moves were observed (they belong to S_+), but any extension of its movements (i.e. one-step longer rides) are in $S_?$. We have the following.

Theorem 1. *If $\mathcal{Q}$ is a limited rider in a game $\mathcal{G}$, and $\mathcal{D}$ a training data described above, then the DFA returned by the [4, Algorithm 2] generates illegal moves.*

Theorem 2. *Consider* single move known *scenario and a game $\mathcal{G}$ containing a limited rider $\mathcal{Q}$ and another piece $\mathcal{R}$ with unlimited ride, i.e. satisfying $\{a_i^j, a_i^{j-1}b_i\} \subseteq L_{\mathcal{R}} \cup S_0$ for all $j > 0$. Let training data $\mathcal{D}$ be such that all the moves of the form a_i^j and $a_i^{j-1}b_i$ are observed for the piece $\mathcal{R}$. Then, the DFA returned by [4, Algorithm 2] generates illegal moves.*

(Proofs of these results are presented in [31]). The practical consequence of Theorem 2 is that given any limited rider piece occurring next to a similar not limited rider (e.g. Short Rook and Rook, Lion Dog and Queen) the learned rules of these figures can be indistinguishable.

The problem addressed in Theorems 1 and 2 lies mainly in the construction of the provided training data, which is hard to detect and handle on the algorithm's side. However, by careful designing of learning and checking procedures, we should be able to guarantee more safety, and more intuitive restrictions on the produced DFA. We will present approaches that try to fix the problem from two sides.

First, we establish dependencies between actually discovered legal moves and hypothesis concerning additional moves from $S_?$. By that, we add additional safeguard and can restrict consistency checking to discard „highly improbable" candidates even if they are consistent with the given data. Secondly, we force consistency checking function to check only those automata we strongly believe they may be good, by using a proper automata learning algorithm, based on more sophisticated heuristic strategy of merging states.

C. RPNI

An alternative solution is to use one of the existing polynomial algorithms for identification DFA's from samples, e.g. Gold [21] or RPNI [24]. Due to its better performance ([26]), we have chosen RPNI as our test algorithm for boardgame move learning.

The arguments of this algorithm are the set of positive samples S_+ and the set of negative samples S_-. Initially, a prefix tree acceptor based on S_+ is constructed. The algorithm searches for a pair of states, such that after merging these states the automaton does not accept any word from S_-. The states are chosen so that one of them is a root of a subtree of the original PTA. The merging procedure disconnects this subtree, merges the states, and then folds the subtree into the constructed DFA so that the resulting DFA remains deterministic. The complexity of the procedure is linear in the size of the folded subtree. For the detailed description, the reader is referred to [12, Section 12.4].

Given the sets of positive samples S_+ and negative samples S_-, the time complexity of RPNI is $O((||S_+|| + ||S_-||)||S_+||^2)$. However, considering the application to Simplified Boardgames and the estimation given in Observation 1, this complexity is unpractical. For this reason, we have to modify consistency checking part of the algorithm from simple iteration through S_- set to some more complex function (e.g. the one used in Björnsson's algorithm). Then we have

Observation 3. *Let S_+ be the language accepted by the piece's prefix tree acceptor pt, and $\mathcal{C}(k, td)$ the complexity of consistency check given training data td and DFA with k states. Then, the complexity of RPNI algorithm is*

$$O((||S_+|| + \mathcal{C}(||S_+||, td))||S_+||^2).$$

The algorithm remains polynomial in the size of the initial prefix tree acceptor, yet again the dominant part depends on the construction of the consistency checking procedure. For that reason, in the next section, we propose alternative procedures focused on efficiency.

IV. Efficient Consistency Checking

Our definition of consistency is that a language has to contain all positive samples (S_+) and reject the negative ones (S_-). In the case of boardgame movements learning, there are many interpretations of what the negative sample is. If we are

able to observe all legal moves in any position, every unlisted move fitting within board has to be considered as negative. All the other moves that have not been rejected, can be labeled in any way.

This is the same problem (of fitting into an unknown target language) like in the standard language inference problem, yet here we know the set S_0 that does not matter at all. Also, we should be able to predict correct and incorrect moves basing on our boardgame related intuition.

A. Fast Consistency Check

Assume the scenario when our priority is to ensure our algorithm learns only correct moves, i.e. we treat $S_?$ in the same way as S_-. We are looking for the language L such that

$$S_+ \subseteq L \quad \text{and} \quad L \cap (S_- \cup S_?) = \emptyset, \tag{1}$$

and the minimal DFA representing L has the smallest number of states.

For the given prefix tree acceptor T defining S_+ and DFA $\mathcal{A}$ approximating L, the optimal procedure checking the consistency of $\mathcal{A}$ is described as Algorithm 1. Starting with the initial state of $\mathcal{A}$, and the root of T, we traverse through $\mathcal{A}$, simultaneously matching visited states with the states in T. The algorithm returns *False* if there is a mismatch in the state acceptance within the T or there is an accepting state outside T but within the board.

Algorithm 1 FastCheck($\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F\rangle$, $x \in Q$, $T = \langle Q', \Sigma, \delta', q'_0, F'\rangle$, $x' \in Q'$, $w \in \Sigma^*$)

```
1:  for all a ∈ Σ do
2:    if ∃y, y'. δ(x, a) = y ∧ δ'(x', a) = y' then
3:      if F(y) ≠ F'(y') then return False end if
4:      if ¬FastCheck(A, y, T, y', wa) then
5:        return False
6:      end if
7:    end if
8:    if O(wa) then continue end if
9:    if F(δ(x, a)) then return False end if
10:   if ¬FastCheck(A, δ(x, a), T, null, wa) then
11:     return False
12:   end if
13: end for
14: return True
```

Let as notice, that in the case of Simplified Boardgames, complexity of the $\mathcal{O}$ function is additive, i.e. given w_1, w_2 and intermediate result of $\mathcal{O}(w_1)$, the value of $\mathcal{O}(w_1 + w_2)$ can be computed in time $O(|w_2|)$. The conclusion is that the check in line 7 can be done in $O(1)$. So the runtime of the algorithm is linear in $||L(\mathcal{A}) \cap (S_+ \cup S_?)||$, which is the upper bound for the worst case complexity from Observation 1.

B. Fractional Acceptance Consistency Check

Another reasonable assumption is that if a DFA representing a language of piece movements is small and does not contradict given data, then with a high probability it is correct. This may not be entirely true when taking into account artificially generated rules, yet in the vast majority, fairy chess pieces can be represented by automata with a simple construction.

Basing on that, and assuming that we have „big enough" training data, we can easily extend *FastCheck* algorithm to allow some fraction of moves from $S_?$. For our *FractionalCheck*$_\alpha$ algorithm we assume that if it produces language L, then at most $(1-\alpha)|L \setminus S_0|$ generated words belong to $S_?$.

The consequence of this assumption is that if L_p is the language of legal moves and S_+ observed valid moves then, given $\frac{|S_+|}{|L_p|} \geq \alpha$, the consistency check will allow optimal DFA despite acceptance of moves from $S_?$. On the other hand, if the sample is small and $\frac{|S_+|}{|L_p|} < \alpha$, the optimal automata will be rejected, as the evidence supporting its correctness are judged as too weak.

The procedure of *FractionalCheck*$_\alpha$ can be described by comparing with *FastCheck* as follows. Instead of returning *False* in case of finding an accepted move from $S_?$, the algorithm has to track the number of such moves. If their number exceeds an established threshold, then the function has to return *False*, at best finishing immediately, e.g. using the exception mechanism. In the *all legal moves* scenario, the additional consistency check with S_- is necessary. This requires browsing through all the observed positions and subset checking between observed and DFA-generated moves (which is equivalent to [4, Algorithm 1], lines 7–8).

V. Spine Compaction Algorithm

Both RPNI and Björnsson's LearnDFA algorithms are in fact general purpose methods, i.e. they do not benefit in any way from the fact that they are applied to learning boardgame piece rules. We would like to present our approach that, in contrast, is entirely based on the assumption that it has to learn rules of the chess-like piece. The goal of this algorithm is fast learning of probable piece movements by performing only specialized state merges and thus minimizing the number of required consistency checks.

The main idea uses the observation that piece PTA's usually have a form of multiple *spines* with similar subtrees attached (see Figure 2 to examine a rook-like piece example). Actual cases may be more complicated (e.g. spine's cycle period greater than one), yet the idea remains similar, also in the cases such as Checkers *man* piece.

Formally, given a (partial function based) DFA $\mathcal{A}$ and a word $w \in \Sigma^*$, we define a *w-spine* as a path in $\mathcal{A}$ starting in some state q and going along the longest prefix v of w^* that determines a valid path in $\mathcal{A}$. In such a case w is called the *vertebra* of the spine, $|w|$ its *period*, and $|v|$ its length. Each spine is encoded in the triple $(vertebra, initial\ state, length)$ (see Figure 3). We are not interested in spines where $|v| < |w|$, and discard them as not proper.

The main procedure (Algorithm 2) consists of two crucial parts. The first one is responsible for finding all spines in a given prefix tree automata. The other one analyzes the obtained spines in proper order and finds the pairs of states being

candidates to merge. First, we describe both these procedures and then present a detailed description of the outline algorithm.

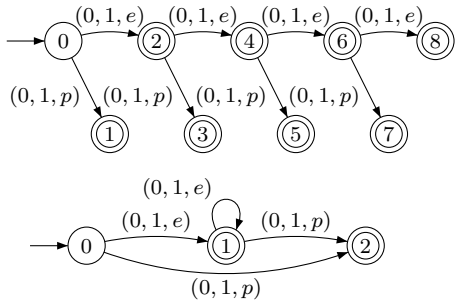


Fig. 2. An example of a chess-like piece sliding only forward. On the top the prefix tree acceptor, on the bottom the optimal DFA for this piece (assuming board of height 5).

A. Spines Selection and Pairs Selection

Given a prefix tree acceptor T, the task of the *spine selection* is to find all proper spines with a period k. Starting from the root r and traversing T using depth-first order, reaching depth k provides the first candidate w for a vertebra. However, if $F(r) \neq F(\delta(r, w))$, then we drop the first letter of w and continue searching for a proper vertebra. Otherwise, we have to analyze outgoing edges. If one is labeled by the first letter of w (say, a), and $F(\delta(r, a)) \neq F(\delta(r, wa))$ we mark $(w, r, |w|)$ as a proper spine, and start a new search from the current node. If state acceptances match, we can continue extending the current spine as long as the traversed path matches w^*. For any other edge labeled by $b \neq a$, we create w' by dropping the first letter of w and appending b at the end and proceed further down with w' as a new vertebra candidate.

The procedure works recursively for a given state considered as the actual root, candidate for vertebra, and length of the longest spine matching so far. Example of the outcome of the given procedure is shown in Figure 3. The time complexity is the same as in the case of the standard DFS, i.e. $\Theta(|Q|)$ given that we traverse only trees.

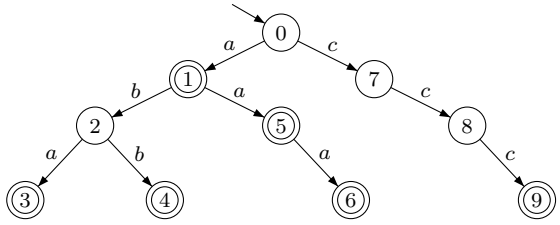


Fig. 3. For the given PTA, the spine selection procedure returns spines as $(vertebra, initial\ state, length)$ triples. Result for $k = 1$: $(a, 1, 2)$, $(c, 0, 2)$. Result for $k = 2$: $(ab, 0, 3)$, $(bb, 1, 2)$, $(aa, 1, 2)$, $(cc, 0, 2)$.

Given a w-spine, the next task is to choose a pair of states (within geodesic distance $|w|$), which will be the best candidate for merging. Our goal is to minimize a chance that the merge result will be rejected by a consistency checking procedure and simultaneously maximize the state reduction.

Let q and q' be two corresponding states in a spine (i.e. the length of the path from q to q' is a multiple of $|w|$), with outgoing vertebra edges labeled by a. In theory, it is safe to merge q and q' if the subtrees rooted in these states are equal except branches initiated with a. If the property is fulfilled for every two corresponding states in a spine, such merge does not add any new words to the language, except for the words longer then v.

In practice, it is enough to have this property only approximately fulfilled. First of all, it is too costly to check the equality of two subtrees. Instead, we check only the first level equality, i.e. if for every letter $b \neq a$, $(q, b) \in \mathrm{Dom}(\delta) \Leftrightarrow (q', b) \in \mathrm{Dom}(\delta) \wedge F(\delta(q, b)) \Leftrightarrow F(\delta(q', b))$. Moreover, we should be aware that given data might be sparse, and it is less probable that we will have valid data concerning longer moves. Given that, we apply a heuristic that allows corresponding subtrees more distance from the spine root to be smaller.

The overall *pair selection* procedure starts with a spine root r as a candidate for merging with $\delta(r, w)$. Then, we traverse through the spine comparing corresponding subtrees and replace candidate for merging if we meet a larger corresponding subtree. Final candidate, if the spine length remains longer then the vertebra, is the base for the selected pair. Figure 4 presents the visualization of the process. The complexity of the procedure is $\Theta(|v| \cdot d)$, assuming d is the maximal degree of a node in a spine.

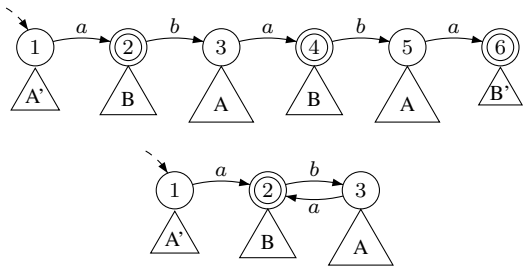


Fig. 4. Example of the spine compaction process. On the top ab-spine rooted in 1, with subtrees such that $A' \subset A$ and $B' \subset B$. On the bottom the situation after compaction: pair $(1, 3)$ is not a safe candidate for merging, so the states 2 and 4 were selected and merged instead.

B. The Main Algorithm

The main part of the Spine Compaction procedure is presented as Algorithm 2. It uses a constant K indicating, similarly as in Björnsson's algorithm, the maximal allowed length of the cycle. After constructing initial prefix tree acceptor, in lines 2–7 we search for all spines not exceeding period K, starting from the children of the root. Thus, we explicitly exclude PTA root for being selected as a spine root, to prevent it from being a part of a cycle (which is an assumption supported by our experiments). In lines 8–11, we investigate each spine to select pairs for further merging.

What remains, is to try to merge every pair and check for consistency (lines 13–18). Very important is the order of applying merge operations. Our strategy is to compact spines with shorter vertebra first, starting with the longest spines.

The merging function (line 15) is a deterministic merge used in RPNI algorithm. At this moment, we operate on states being the sets of original states, to be able to track merging process. We maintain a set of pairs that resulted in inconsistent automata (lines 12, 17), which is used to skip unnecessarily repeated computations (line 14). Additionally, we prevent pairs to create a multiloop, which is an experience-based heuristic, as fairy-chess pieces representations rarely have one. To check this condition it is enough to analyze δ values for the states in current pair. Finally, we minimize the resulting automaton.

Observation 4. *Let S_+ be the language accepted by the piece's PTA pt, and $\mathcal{C}(k, td)$ the complexity of consistency check given training data td and DFA with k states. We can estimate upper bound on the number of spines selected in Algorithm 2 by $O(||S_+||^2)$. Then, the complexity of the SpineCompaction algorithm is*

$$O((||S_+||^2 + \mathcal{C}(||S_+||, td))||S_+||^2).$$

Algorithm 2 SpineCompaction(Piece pt, TrainingData td)

1: $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F\rangle \leftarrow constructPTA(pt, td)$
2: $spines \leftarrow \emptyset$
3: **for all** $a \in \Sigma$ if $\langle q_0, a\rangle \in \mathrm{Dom}(\delta)$ **do**
4: **for** $k \leftarrow 1$ to K **do**
5: $spines \leftarrow spines \cup \mathcal{A}.SelectSpines(k, \delta(q_0, a), \varepsilon, 0)$
6: **end for**
7: **end for**
8: $pairs \leftarrow \emptyset$
9: **for all** $s \in spines$ **do**
10: $pairs \leftarrow pairs \cup \{\mathcal{A}.SelectPair(s)\}$
11: **end for**
12: $forbidden \leftarrow \emptyset$
13: **for all** $p \in pairs.OrderByPriority()$ **do**
14: **if** $p \in$ *forbidden* $\vee$ *isMultiloop* **then continue end if**
15: $\mathcal{A}' \leftarrow MergeAndFold(\mathcal{A}, p)$
16: **if** $consistent(pt, \mathcal{A}', td)$ **then** $\mathcal{A} \leftarrow \mathcal{A}'$
17: **else** $forbidden \leftarrow forbidden \cup \{p\}$ **end if**
18: **end for**
19: **return** $minimize(\mathcal{A})$

VI. Experiments and Empirical Evaluation

We have performed experiments to compare the three presented learning algorithms paired with different consistency check functions in both the *single move known* and *all legal moves* scenarios.

For the *all legal moves* scenario we have prepared 20 datasets per game, each containing record of 50 plays generated using two random agents playing against each other. In the case of the *single move* scenario, the number of play records in each dataset was increased to 1000. The maximum game length was set to 80 moves per player in all cases. Due to the symmetry of games, we have performed learning only for the white player (results presented in [4], [15] show no significant difference for white and black in the case of such games). If not stated otherwise, the *MaxExpansions* constant of Björnsson's algorithm was set to 20, and K used in Björnsson's and Spine Compaction algorithms was set to 2. All experiments were run on 2.60GHz Intel Core i7-6700HQ processor.

TABLE I
Experiment results for the *all legal moves* scenario.

Consistency Check	Correct size (%)			Errors (%)		
	Bjö.	RPNI	SC.	Bjö.	RPNI	SC.
Björnsson	87.6	**91.5**	87.3	**4.9**	9.1	7.6
Fractional$_{0.5}$	87.6	**89.9**	84.9	**4.9**	6.3	7.6
Fractional$_{0.6}$	89.0	**89.8**	86.6	**2.4**	4.0	5.0
Fractional$_{0.7}$	**87.1**	**87.1**	**87.1**	**2.4**	3.5	3.2
Fractional$_{0.8}$	**85.6**	85.5	**85.6**	**2.4**	3.7	3.2
Fractional$_{0.9}$	**73.7**	73.5	**73.7**	**2.4**	2.7	3.9
Fast	**69.3**	**69.3**	**69.3**	**0**	**0**	**0**

A. Games

We have used as a testbed 8 fairy chess games, containing 41 pieces altogether. The chosen games can be divided into three categories. The first one consists of simplified versions of known boardgames or their variants. It includes *chess*, *Los Alamos* (small chess variants without bishops), *Tamerlane* (large 10×11 game containing many non-standard pieces like giraffe, camel, picket), and a breakthrough variant of *Checkers*. (See [32], [4] for more detailed game descriptions.)

The other three games are the result of procedural content generation algorithms [33], [34]. Thus, they all contain only non-standard pieces. The last game was handcrafted especially to fool unaware learning algorithms. It contains two special pieces that can move horizontally like a rook without capturing moves, yet with limited movement lengths. The initial position is constructed such that every move exceeding these lengths is impossible due to the obstacles. Thus, the setup fulfills prerequisites given in the Section III-B, and there is no evidence of nonconsistency when assuming the $(1, 0, e)^* + (-1, 0, e)^*$ language for these pieces.

B. Results and Conclusions

Tables I and II show the results for the *all legal moves* and *single move known* scenarios (Spine Compaction algorithm is abbreviated as SC). For every *(learning algorithm, consistency check)* pair, we have computed the percent of generated automata with the optimal size (which is the task of regular language inference), and the percent of automata generating consistency *errors*. By an *error*, in this context, we mean a non-empty intersection with S_- of the piece's true language. These data visualize the trade-off between the rapid language expansion followed by the DFA size reduction and performing only minor adjustments to the initial prefix tree acceptor. Runtimes of all experiments are presented in Table III.

The experiments show that our *SpineCompaction* learning algorithm provides most accurate results in the *single move known* scenario of learning, and comparable although more erroneous results in the GGP-like *all legal moves known* scenario. Moreover, it requires much less time for learning than

TABLE II
EXPERIMENT RESULTS FOR THE *single move known* SCENARIO.

Consistency Check	Correct size (%)			Errors (%)		
	Bjö.	RPNI	SC.	Bjö.	RPNI	SC.
Björnsson	**73.7**	70.6	**73.7**	**5.4**	19.8	9.6
$Fractional_{0.6}$	**88.7**	61.5	87.3	6.8	40.4	**6.1**
$Fractional_{0.7}$	**89.9**	65.7	89.8	4.9	37.6	**2.8**
$Fractional_{0.8}$	**88.0**	64.6	**88.0**	4.5	34.0	**4.4**
$Fractional_{0.9}$	**73.3**	71.5	**73.3**	3.7	14.4	**3.4**
Fast	**68.5**	**68.5**	**68.5**	**0**	**0**	**0**

TABLE III
LEARNING TIMES (IN SECONDS).

Consistency Check	*all legal moves*			*single move known*		
	Bjö.	RPNI	SC.	Bjö.	RPNI	SC.
Björnsson	68.2	16.4	**4.1**	1598.6	454.1	**82.4**
$Fractional_{0.5}$	51.8	9.4	**2.2**			
$Fractional_{0.6}$	50.8	9.3	**2.1**	5.4	4.4	**0.2**
$Fractional_{0.7}$	50.8	9.2	**2.0**	5.6	3.3	**0.2**
$Fractional_{0.8}$	49.4	9.5	**1.8**	5.7	3.0	**0.2**
$Fractional_{0.9}$	42.8	10.4	**1.5**	5.7	2.6	**0.2**
Fast	4.0	4.6	**0.4**	4.6	5.3	**0.4**

the other tested algorithms. Experiments show that in practice gathered observations data are incomplete, which justifies the need for a learning function based on some approximation method. For this reason, we have also proposed consistency check functions, determining the acceptance of the proposed DFA's on the basis of the fraction of unsafe moves. These are $FractionalCheck_\alpha$ and *FastCheck* (which is a special case of *FractionalCheck* with $\alpha = 1$). As Tables I and II show, we can select an α value giving us better results then Björnsson's consistency check, which has been our reference point.

Nevertheless, all the tested methods are efficient in comparison to game reasoners used in GGP. This supports the thesis that the problem of General Game Playing should be solved by detecting subclasses with more effective, domain-based algorithms applicable [13]. As the boardgames are one of the most common classes of games used in GGP, the effort in this direction is relevant for the domain, and has possible practical applications in the improvement of GGP systems [4].

ACKNOWLEDGMENTS

This work was supported in part by the National Science Centre, Poland, under the project numbers 2017/25/B/ST6/01920 (Jakub Kowalski) and 2015/17/B/ST6/01893 (Andrzej Kisielewicz).

REFERENCES

[1] M. Genesereth and M. Thielscher, *General Game Playing*. Morgan & Claypool, 2014.
[2] M. Genesereth, N. Love, and B. Pell, "General Game Playing: Overview of the AAAI Competition," *AI Magazine*, vol. 26, pp. 62–72, 2005.
[3] N. Love, T. Hinrichs, D. Haley, E. Schkufza, and M. Genesereth, "General Game Playing: Game Description Language Specification," Stanford Logic Group, Tech. Rep., 2006.
[4] Y. Björnsson, "Learning Rules of Simplified Boardgames by Observing," in *ECAI*, ser. FAIA, 2012, vol. 242, pp. 175–180.
[5] J. Kowalski and A. Kisielewicz, "Towards a Real-time Game Description Language," in *ICAART*, vol. 2, 2016, pp. 494–499.
[6] T. Schaul, "A video game description language for model-based or interactive learning," in *IEEE CIG*, 2013, pp. 1–8.
[7] M. Thielscher, "A General Game Description Language for Incomplete Information Games," in *AAAI*, 2010, pp. 994–999.
[8] D. Perez, S. Samothrakis, J. Togelius, T. Schaul, and S. M. Lucas, "General Video Game AI: Competition, Challenges and Opportunities," in *AAAI*, 2016, pp. 4335–4337.
[9] M. Świechowski, H. Park, J. Mańdziuk, and K. Kim, "Recent Advances in General Game Playing," *The Scientific World Journal*, vol. 2015, 2015.
[10] B. Pell, "METAGAME in Symmetric Chess-Like Games," in *Heuristic Programming in Artificial Intelligence: The Third Computer Olympiad.*, 1992.
[11] J. Pitrat, "Realization of a general game-playing program," in *IFIP Congress*, 1968, pp. 1570–1574.
[12] C. de la Higuera, *Grammatical Inference: Learning Automata and Grammars*. Cambridge University Press, 2010.
[13] J. Kowalski and A. Kisielewicz, "Testing General Game Players Against a Simplified Boardgames Player Using Temporal-difference Learning," in *IEEE Congress on Evolutionary Computation*, 2015, pp. 1466–1473.
[14] J. Kowalski, J. Sutowicz, and M. Szykuła, "Simplified Boardgames," 2016, arXiv:1606.02645 [cs.AI].
[15] P. Gregory, Y. Björnsson, and S. Schiffel, "The GRL System: Learning Board Game Rules With Piece-Move Interactions," in *IJCAI Workshop on General Intelligence in Game-Playing Agents*, 2015, pp. 55–62.
[16] S. Cresswell and P. Gregory, "Generalised Domain Model Acquisition from Action Traces," in *International Conference on Automated Planning and Scheduling*, 2011, pp. 42–49.
[17] S. N. Cresswell, T. McCluskey, and M. M. West, "Acquisition of Object-Centred Domain Models from Planning Examples," in *International Conference on Automated Planning and Scheduling*, 2009, pp. 338–341.
[18] S. Muggleton, A. Paes, V. Santos Costa, and G. Zaverucha, "Chess Revision: Acquiring the Rules of Chess Variants through FOL Theory Revision from Examples," in *Inductive Logic Programming*, ser. LNCS, vol. 5989, 2010, pp. 123–130.
[19] Ł. Kaiser, "Learning Games from Videos Guided by Descriptive Complexity," in *AAAI*, 2012, pp. 963–969.
[20] J. Kirk and J. E. Laird, "Interactive task learning for simple games," *Advances in Cognitive Systems*, vol. 3, pp. 11–28, 2014.
[21] E. M. Gold, "Complexity of automaton identification from given data," *Information and Control*, vol. 37, no. 3, pp. 302–320, 1978.
[22] L. Pitt and M. K. Warmuth, "The Minimum Consistent DFA Problem Cannot Be Approximated Within Any Polynomial," *Journal of the Association for Computing Machinery*, vol. 40, no. 1, pp. 95–142, 1993.
[23] B. A. Trakhtenbrot and Y. M. Barzdin, *Finite automata: Behavior and Synthesis*. American Elsevier Publishing Company, 1973.
[24] J. Oncina and P. Garcia, "Identifying Regular Languages In Polynomial Time," in *Pattern Recognition and Image Analysis*, 1992, pp. 49–61.
[25] K. Lang, "Random DFA's can be Approximately Learned from Sparse Uniform Examples," in *Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory*, 1992, pp. 45–52.
[26] P. García, A. Cano, and J. Ruiz, "A Comparative Study of Two Algorithms for Automata Identification," in *Grammatical Inference: Algorithms and Applications*, ser. LNCS, 2000, vol. 1891, pp. 115–126.
[27] P. Dupont, "Incremental regular inference," in *Grammatical Interference: Learning Syntax from Sentences*, ser. LNCS, 1996, vol. 1147, pp. 222–237.
[28] A. Cano, J. Ruiz, and P. García, "Inferring Subclasses of Regular Languages Faster Using RPNI and Forbidden Configurations," in *Grammatical Inference: Algorithms and Applications*, ser. LNCS, 2002, vol. 2484, pp. 28–36.
[29] R. Parekh and V. Honavar, "Learning DFA from Simple Examples," *Machine Learning*, vol. 44, no. 1, pp. 9–35, 2001.
[30] Wikipedia. (2017, January) Fairy chess piece — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/Fairy_chess_piece.
[31] J. Kowalski, "General Game Description Languages," Ph.D. dissertation, University of Wrocław, 2016.
[32] (2016) The Chess Variant Pages. http://www.chessvariants.org/.
[33] J. Kowalski and M. Szykuła, "Procedural Content Generation for GDL Descriptions of Simplified Boardgames," 2015, arXiv:1108.1494 [cs.AI].
[34] ——, "Evolving Chesslike Games Using Relative Algorithm Performance Profiles," in *Applications of Evolutionary Computation*, ser. LNCS, 2016, vol. 9597, pp. 574–589.

Strategic Features and Terrain Generation for Balanced Heroes of Might and Magic III Maps

Jakub Kowalski, Radosław Miernik, Piotr Pytlik, Maciej Pawlikowski, Krzysztof Piecuch, Jakub Sękowski
Institute of Computer Science, *University of Wrocław*
Wrocław, Poland
jko@cs.uni.wroc.pl, radekmie@gmail.com, piotrcpytlik@gmail.com,
pawlikowski.maciek@gmail.com, kpiecuch@cs.uni.wroc.pl, sequba@gmail.com

***Abstract*—This paper presents an initial approach to a generic algorithm for constructing balanced multiplayer maps for strategy games. It focuses on the placement of so-called *strategic features* – map objects that have a crucial impact on gameplay, usually providing benefits for the players who control them.**

The algorithm begins with constructing a logical layout of the map from the perspective of a single player. We use a novel approach based on graph grammars, where rules do not add new features but are initially constrained by the content of the start node, which simplifies their construction. To introduce other players' space, the single-player graph is multiplied and partially merged. The result is projected onto a grid by combining Sammon mapping, Bresenham's algorithm, and Voronoi diagrams. Finally, strategic features are arranged on the map using an evolutionary search algorithm.

We implemented the proposed method in a map generator which we are developing for the popular strategy game Heroes of Might and Magic III. We also point out how this approach can be adapted to other games, e.g. StarCraft, WarCraft, or Anno.

***Index Terms*—procedural content generation, map generation, graph grammars, heroes of might and magic**

I. Introduction

Procedural content generation (PCG) has been used in games since the introduction of home computers. Examples include Beneath Apple Manor from 1978, or released in 1980 Rogue. However, the progress of PCG techniques did not go hand in hand with the progress in other areas of game development. Nowadays, most popular commercial video games, especially AAA titles, either do not use procedural generation or use relatively simple constructive generative techniques [1], which are not especially interesting from the research point of view.

World generation in Civilization VI [2] is based mainly on Perlin noise and simple heuristics for objects placement. Dungeons in Diablo III [3] consist of handcrafted tiles that are combined together using a simple generator, and later populated with randomized treasures and monsters. The levels in Spelunky [4] always contain 16 rooms whose layouts are selected from a set of predefined templates. To increase visual diversity, random obstacles are placed according to the templates' indications. Lastly, monsters are put in spots determined by heuristic rules.

On the other hand, academia-rooted PCG research often focuses on more advanced techniques, such as evolutionary algorithms, simulation-based evaluations, or neural networks [5]. However, their implementations usually have a purely scientific character, as they are applied mostly to test-bed games and tools [6], [7], [8], [9], [10], [11], [12], [13], indie games [14], or reimplementations of older titles [15], [16], [17]. There are, of course, notable exceptions, including level generation for Angry Birds [18], Cube 2 [19], [20], or StarCraft [21]. Some academic approaches even turned out into commercial (usually independently published) games, e.g. Galactic Arms Race [22], Petalz [23], or Darwin's Demons [24]. Nevertheless, it is often a difficult and laborious task to meet all of the requirements and generate maps that are playable in an already existing, commercially published game.

Our ambition is to develop a map generator for Heroes of Might and Magic III (HoMM3) [25] which could be widely used by the game's community. This paper describes the first step towards this goal. We are presenting an algorithm that generates a balanced map layout, projects it onto a grid, and ensures a fair placement of the most important map features, mainly towns and mines.

One of the main reasons for our work is, as we believe, a community need for such a tool. Although the game is nearly 20 years old, it is considered the best part of the entire Heroes of Might and Magic series and is still widely played. There are multiple fan-made mods and extensions, including Horn of the Abyss, In the Wake of Gods, and VCMI – an attempt to entirely rewrite the original HoMM3 engine. Additionally, in 2015, Ubisoft released the HD Edition of the game ($642,075 \pm 24,614$ sold copies according to the Steam Spy website[1]). Sadly, this version does not include expansions and lacks the random map generator.

Existing template-based generators usually require a skilled user to modify generator's behavior. Instead, we aim to propose a method that will be able to produce reliable maps based on simple user preferences. We provide necessary background about map generation approaches in the next section. Our algorithm is fully described in Section III. Section IV contains details of our implementation, which is tailored for Heroes of Might and Magic III and yields playable maps. We discuss the results produced by our algorithm, present the arguments for

[1]https://steamspy.com/app/297000, retrieved March, 2018

978-1-5386-4359-4/18/$31.00 ©2018 IEEE

its generality, and give a perspective of the future research in Section V. Finally, we conclude in Section VI.

II. Related Work

A. Procedural Map Generation in Games

In this section, we provide a brief overview of the PCG research and algorithms relevant to this study. For the comprehensive survey on procedural generation and its role in games, we refer to textbooks [5], [26].

Graph grammars ([27], [28]) allow to combine a powerful and intuitive concept of grammatical rules with a capacious graph representation. This approach is often used for creating non-linear story-driven levels, where the level structure strongly follows the generated plot. In other words, all possible paths towards the objectives, along with the events that can be encountered along the way, are constructed as a randomly generated graph.

In general, application of graph grammar rules is a complicated task, as the left-hand part of a rule has to match a proper subgraph, which then has to be modified in-place according to its right-hand side. To make generated structures more predictable, additional constraints are usually placed upon the maximum number of application of certain rules or resources that those rules can place. This causes the entire architecture to become significantly more complex.

Graph grammars are especially convenient for generating RPG-style one-player maps. A good example is a mission graphs generator for Dwarf Quest [14]. It is based on evolutionary search, where grammatical rules are applied as mutation operators. Evolved mission graphs are converted into Dwarf Quest levels consisting of pre-made rooms mapped to nodes using so-called layout solver. Another approach, presented in [12], generates Zelda-like levels via two-step evolutionary algorithm: the first phase generates an acyclic high-level graph of areas, while the second evolves, for each area, a low-level cyclic graph consisting of rooms and doors.

An important issue is mapping the abstract graph structure into concrete locations while preserving its characteristics. In [29], the problem has been restrained to mapping trees into a low-resolution grid (one node per square) and solved via a recursive backtracking algorithm. If no mapping solution has been found, the tree is discarded and regenerated.

An approach of generating RTS-style maps via multiobjective evaluation has been presented in [6]. The method had since been extended and used to generate playable maps for StarCraft, however, it commonly produces maps "not looking very StarCraft-like" [21]. The genotype consists of player base locations, resource (gas and minerals) locations, and information about impassable areas. The algorithm tries to optimize maps mainly for their fairness (various resource-related metrics like distance, ownership estimation, safety) and strategic aspects (metrics related to choke points and possible unit paths). Recently, a new idea of using neural networks to predict the placement of StarCraft map features based on the existing map data has been proposed and tested in [30].

Lastly, we want to raise the subject of using cellular automata (CA) as a map-filling algorithm. It is often needed to populate an area with obstacles (water, impassable rocks, etc.) in a way that is random, fast, and still somewhat controllable. In [31], cellular automata have been utilized to generate surprisingly lifelike cave-rooms. In particular, the authors perform an exhaustive study on the influence of the CA parameters on the style of the obtained caves. In the genetic algorithm for generating Dune 2-like maps [17], an interesting approach to cellular automata-based genotype-to-phenotype mapping is used. Instead of direct map representation, the evolutionary process modifies only the CA parameters (e.g. size of a Moore-neighbourhood, activation threshold, number of iterations), which are later used to fill the map with rock, sand, and indirectly other features like player starting zones.

B. HoMM3 Map Generation

A random map generator had been introduced into HoMM3 with the first expansion – Armageddon's Blade. Despite many faults it was, and still is, commonly used (its exclusion from HoMM3 HD Edition caused many complaints from the players). The generator on its default settings tends to produce highly unbalanced maps filled with items that are rarely used by human map-makers (e.g. Pandora's Boxes with very random rewards), and valuable treasures that often lie unprotected. One of its bugs can even occasionally make a player starting position completely surrounded by obstacles.

However, by overriding generator's template, it is possible to modify, to some limited extent, the algorithm's behavior. This allows to handcraft a structure similar to our map layout (see Section III-C), which contains information about the zones' characteristics and relative positions. Many alternative templates have been made and shared among the community[2].

The improvements of the random map generator are important aspects of HoMM3 mods. Horn of the Abyss provides a graphical template editor offering a wide array of settings[3]. Recently, a new web-based map generator had been published[4]. It is a work-in-progress version and still does not support all map features (e.g. water, underground). It defines its own format of templates, which allows for a very high level of customization and gives a lot of control over the resulting maps.

III. Methodology

Let us start with the definitions. A *zone* represents consistent map area with the same purpose, style, and level of challenge. Zone is defined by its *class*, i.e. (*type*, *level*) pair. The higher the level, the more challenging and rewarding the content of the zone should be. The zones are *local* if they are easier accessible for one player, i.e. it should be safe to explore for that player. The *buffer* zones are the areas separating different players' local areas. Buffers are equally accessible by at least

[2]http://forum.heroesworld.ru/showpost.php?p=564869&postcount=57, retrieved March, 2018 (in Russian)

[3]https://www.facebook.com/h3hotaofficial/posts/1447460758611590

[4]http://www.frozenspire.com/MapGenerator/Index.html, retrieved March, 2018

two players, so in these zones the multiplayer fighting should take place. The *goal* zone is a special buffer, limited to one per map, existing only in maps with specific winning conditions (capture town, defeat monster, acquire artifact).

The content of zones is represented by *features*. There are strategic features, like *town* and *mine*, or more technical features: *outers* and *teleports*. The *value* of a feature describes its detailed content. We distinguish player's main town, other towns they initially possess, towns dependent on surrounding factions, and two types of random towns (chosen either by our generator or in-game randomizer). There are *base* mines (for wood and ore), *primary* mines (for faction-dependent most important resources), *gold* mines, and *random* mines.

Outer represents a connection with other player's part of the map. Teleport is a special kind of outer: a hyperedge joining all of its occurrences within the map via two-way monoliths. The value of an outer is its level – representing difficulty in the same way as levels of zones and influencing the strength of guarding creatures. The value of a teleport is its level and identifier. Teleports with the same identifier are joined together and there can be at most 4 distinct teleports on the map.

A. Generation Parameters

First, the user needs to specify desired map characteristics. Apart from necessary settings like map size, players' specification, or winning condition, we defined eight parameters influencing generator's behavior. All the parameters have values from 1 to 5, where the default 3 means "standard". The most important for the algorithm are:

- *welfare* – higher values mean more resources and mines,
- *towns* – higher values mean more towns placed,
- *branching* – higher values mean more connections between zones,
- *focus* – lower values mean more player-vs-player map, while higher values focus on player-vs-environment,
- *zonesize* – higher values mean bigger zone areas (which is equivalent to lesser number of zones placed).

Knowing the map specification, we start by defining the content of the map without knowledge about its layout. We do it from the perspective of one player, i.e. we enumerate the zones he will encounter without going into other players' territory. The generator uses a set of parameterized, randomized rules. For instance: number of zones depends on the map size and zonesize parameter, maximum zone level depends on the map size and overall estimated difficulty of the map, and strong player-vs-player focus results in a smaller number of local zones compared to the number of buffers.

Let us start with an example. For M-size, 4-player map generated with default parameters, we can obtain the following zone classes: $(local, 1)$, $(local, 3)$, $(buffer, 2)$, $(buffer, 4)$.

We generate features in a similar way. However, each feature is already associated with a zone class. Although we do not know an exact zone to place a certain feature, we know what class of challenge this feature should belong to. These classes are calculated based on feature values (e.g. gold mines have a higher probability to be placed in high-level zones) and given parameters (low player-vs-player focus forbids putting outers in low-level local zones, winning condition "capture town" requires town in the goal zone, etc.).

Continuing our example, we can obtain a starting town (T_{START}) and two base mines (M_{BASE}) in $(local, 1)$ zone, two random mines (M_{RND}) in $(local, 3)$, a random (T_{NEUT}) town in $(buffer, 4)$, and an outer edge in each buffer. (See Figure 1a.)

B. Logic Map Layout

The *Logic Map Layout* (LML) graph consist of nodes representing zones and edges representing connections between the zones. Each node contains a multiset of zone classes and a multiset of features (with a proper class associated). LML defines a logical structure of the map and is constructed using a novel variant of a graph grammar algorithm.

An LML node is *inconsistent* if it contains a feature associated with a class which is not present in the node. We require our graph to always be consistent. A node is *final* if it is consistent and contains only one zone class.

We initialize our graph structure with one node that consists of all zone classes and features computed from the generation parameters, as shown in Figure 1a. In our algorithm, graph generation process comes down to making all nodes final. Thus, our approach does not require any additional constraints checking, as all we do is a redistribution of the nodes' content.

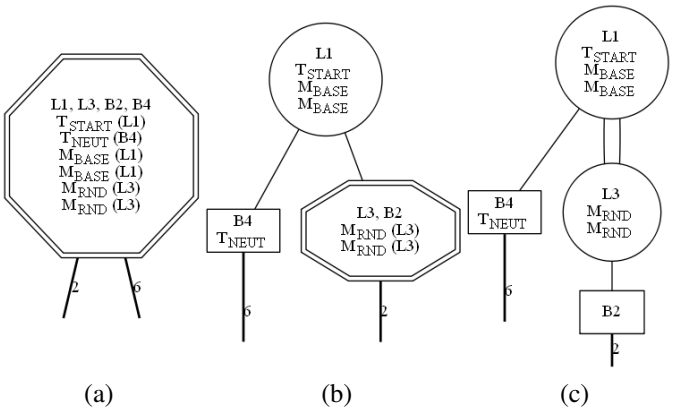


Fig. 1. Constructing LML. Nodes contain classes and features, bolded lines are outer edges with levels. Initial node is presented in Fig. 1a. Fig. 1b contains one of the intermediate steps (after two successful applications of rule (1)). The final LML is shown in Fig. 1c.

Each grammar rule has a weight assigned. In every step, one rule is chosen via the roulette-wheel selection. If its preconditions match, the rule is applied. Otherwise, the graph remains unchanged. Currently, we use four production rules (the weight in parentheses is either a constant or a parameter value):

1) (15) For the first non-final node, divide its content by pushing out a new node containing zones larger than a random pivot.
2) (15) For the first non-final node, if it only contains n zones with the same class, divide its content into n new final nodes and put them at the same depth.

3) (*branching*) Duplicate random edge (if there is only one edge between the nodes).
4) (*branching*) Connect two random, previously not connected, nodes (only local-local or nonlocal-nonlocal).

Although this set can be extended infinitely by adding more and more sophisticated rules, the ones we defined cover most types of reasonable graphs, while remaining relatively simple.

The effectiveness of this grammar comes from a proper ordering of the zone classes. We order zones of the same type by their level, and otherwise we have: local $<$ buffer $<$ goal. This way rule (1) always creates a new node containing a buffer zone if the original node had one. It prevents a local zone from being placed "after" a buffer (counting from the starting position). Rule (1) alone will eventually give us trees where any non-final node contains zones of only one class.

Thus, the rule (2) splits such nodes by making several new nodes all containing only one zone (so they become final). The new nodes have all or only some of the base node's edges (depends on branching parameter). The main issue here is fair features redistribution. We defined heuristic values for every type of feature. Thus, for each feature in the base node, we insert it into the copy that has the lowest value at that moment. The ordering in which the features are considered depends on the zone's type. For local zones, we distribute towns and mines first, while for the other types we prioritize outers and teleports. The role of remaining two rules is to extend otherwise tree-like graphs with cycles and multiedges.

The final LML graph has been presented in Figure 1c. To visualize the process, Figure 1b contains an exemplary middle step, before applying rule (3) to duplicate an edge and rule (1) to the only remaining non-final node.

C. Multiplayer Logic Map Layout

In the next step, we need to compute a layout for the entire map, including all players and all zones. We call the resulting structure *Multiplayer Logic Map Layout* (MLML). To create this graph, we make a copy of LML for each player. Then, we join these duplicated LML's via the outer edges and merge certain buffer zones. In doing so, we want to obtain a graph which is connected and isomorphic from the perspective of every player (i.e. the node containing player's main town).

We say the MLML zones belonging to different players are *corresponding* if they were created as a copy of the same LML zone. We call a newly added edge *valid* when it connects two zones of the same level. First, we attempt to connect all the players' graphs with valid edges between buffer zones, to ensure the final graph will be connected.

After using the buffer zones' outer edges, we add valid edges between local zones, connecting the graph if still needed, and simply distributing them randomly and evenly otherwise. This can be optimized by keeping track of the added edges for each players' corresponding zones and ensuring all players have similar edges. Remembering which corresponding zones had a connection added between them lets us provide a much higher chance for the final graph being isomorphic.

After distributing all outer edges, we merge certain buffer zones, to simplify the graph and allow for a larger buffer zone to replace several corresponding buffer zones. This is done by restricting ourselves to a graph made up of the newly added edges. We search through this graph for sets of corresponding buffer zones, which only have connections between each other.

If such sets exist, each one can be merged into a single buffer zone. While merging such zones, we say the merged zone has a size equal to the sum of sizes of the original buffers, up to a maximum of 3 times the base size (this is an arbitrary limit, which has proven to work well enough during tests). After these merges are finished, we want to verify that the final graph is isomorphic as observed by the players by using a rooted tree isomorphism algorithm[5].

MLML for our example has been presented in Figure 2.

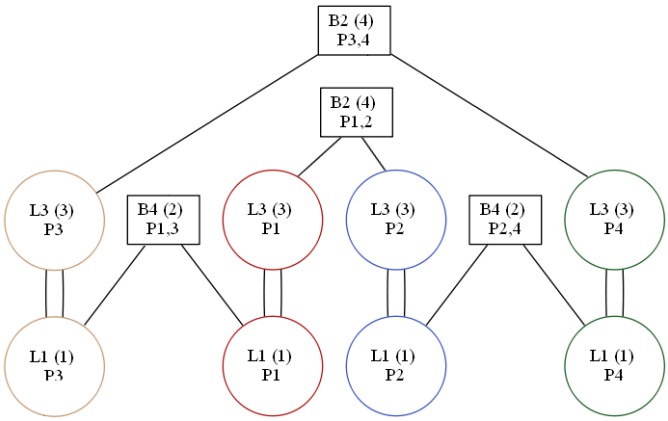


Fig. 2. Generated MLML. Numbers in braces identifies the corresponding LML zones. Each zone contains information about accessing them players.

D. Mapping MLML to Grid

Our task is to project the MLML graph onto a grid, i.e. each vertex has to become a separate zone on a game map. We are bound by three constraints: (a) zones representing connected vertices should be directly accessible from one another, (b) vertex size should represent zone area, and (c) the map should not contain too much unutilized space. We divide this process into two stages. First, we embed the graph points in a planar space. Then, we calculate a modified Voronoi partitioning based on the result. The visualization of each step is presented in Figure 3.

Solving the first constraint algorithmically is not a trivial task. Instead, we chose to pursue a data science approach and employ Sammon mapping [32] for creating a graph embedding. Sammon mapping is a data visualization method deriving from multidimensional scaling. Given a set of points and a matrix d^*_{ij} of relative distances between them, it embeds the points in a low-dimensional space by minimizing a stress function

$$E = \frac{1}{\sum\limits_{i<j} d^*_{ij}} \sum_{i<j} \frac{(d^*_{ij} - d_{ij})^2}{d^*_{ij}},$$

[5]https://groupprops.subwiki.org/wiki/Rooted_tree_isomorphism_problem

where d_{ij} is a distance matrix for the embeddings. Defining a distance between two vertices as a length of the shortest path between them allow us to use this method to draw a graph on a plane [33].

In order to fulfill the (b) requirement, we start with reshaping the graph. We split each vertex of size s into a cycle of s subvertices. Edge (p, q) in the original graph is translated to a random connection between the cycles for p and q. After embedding the obtained graph using Sammon mapping, we still need to ensure the (c) property. Consequently, the output is rotated and cropped. Afterward, we use a gravity-like mechanism to fill sparse regions by pulling the points closer to the map edges.

Note that this method relies on heuristics, and the desired proportions between areas are only roughly maintained. Nevertheless, we found the results satisfying for the task.

At this point, we have a good basis for the partitioning of a map space. To prepare for that, we add a sparse virtual grid above the map grid. This virtual grid covers the map grid and consists of *sectors*, which are rectangles of equal size, each containing a group of map squares.

We call two sectors *direct neighbors* when they are next to each other horizontally or vertically, but not diagonally. For every zone, we want to have a group of sectors, where each one is a direct neighbor of at least one other sector in that zone. This ensures that every zone has a connected set of sectors.

Furthermore, we want every pair of zones which are connected in the MLML graph, to have at least one pair of sectors (one from each zone), which are direct neighbors. This will allow us to later specify that the two zones have an edge between these two sectors.

In our approach, we assign each zone a starting sector, by taking the average of the zone vertices obtained with Sammon mapping and choosing the sector which holds this position.

After assigning each zone a starting sector, we go through the edges from MLML and attempt to connect each pair of zones with a chain of direct neighbors. First, we calculate a path between both starting sectors using a Bresenham algorithm[6]. This ensures that we always have direct neighbours along the way. Next, we traverse the path and try to find a chain of sectors, which start from one of the zones, ends in the other, and has only empty sectors along the way. If such a chain exists we assign the sectors fairly to both zones. So, ultimately, a path starts in one zone, goes along sectors of this zone, and then continues in sectors of the second zone.

Now that we have assigned sectors for all of the zones, we proceed to fill the map grid with a basic Voronoi method. Inside every sector, we generate three random control points with the sector's id. We only allow the points to be generated at a certain distance away from the sector's sides. We assign each grid square the id of the control point which is closest to the center of the square. We only have to take into account the current and neighboring sectors, because further control points can never be closer.

[6]https://www.redblobgames.com/grids/line-drawing.html#stepping

While testing the algorithm, we observed that using too few control points caused neighboring sectors to be separated by another sector. However, when using too many control points, the sector boundaries immediately took the form of a standard grid, without any irregularities. The three random control points allow the Voronoi grid some randomness, while not restricting the sectors to have any specific shape.

Because of the allowed movements in HoMM3, the borders between sectors of different id's can not have diagonal gaps. To decide which grid squares must form a border, we iterate over the squares and compare a candidate to each of its 8 neighbors. When comparing the square with a neighbor, we check if both squares have an id of an assigned sector and if the candidate square has a larger distance to its control point. If it does, it is changed to a wall. Otherwise, it remains unchanged and we continue through the map. The gates between two adjacent zones are placed in zone border tiles such that adjacent squares belong only to one of that zones or are neutral.

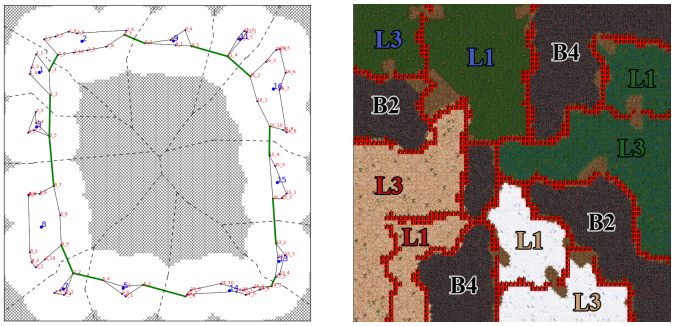


Fig. 3. On the left, result of the Sammon mapping. On the right, final partitioning of a grid, including gates between the appropriate zones. (For this partitioning example, we distorted the graph embedding results to forcefully fill the entire map.)

E. Strategic Features Placement

To guarantee a decent level of balance, we have to place the strategic features such that all players have similar access to the corresponding objects in the corresponding zones. If, for example, one of the players has a mine near their town, while all the others have it on the far end of the zone, this will result in a large difference in their early advancement.

The task can be defined as follows. We are given a set of corresponding zones and, for each of them, coordinates of *entrances*, i.e. tiles where a player can enter the zone. Given a set of strategic features, the following distances should be preserved:

- between corresponding features in corresponding zones (feature-to-feature),
- between corresponding objects and corresponding entrances in corresponding zones (entrance-to-feature),
- if the zone is a buffer, between k-th nearest object for any two entrances in the zone leading from lower-level zones (for k less equal than the number of features).

The first two rules focus on fairness between corresponding zones. The last rule ensures that different players arriving from

their entrances to one particular zone will encounter a strategic feature in a similar distance.

We aim to optimize the features placement via an evolutionary algorithm. A genotype contains the exact position of every feature. We start with computing all valid feature spots in a zone and use them to randomly generate an initial population.

To compute the evaluation function, the distances between all pairs of objects are calculated by BFS. For individuals with overlapping features, the fitness is infinity. Otherwise, we try to minimize the sum of squared differences between corresponding distances included in the above list, e.g. we minimize the discrepancy between the distance from the entrance to the mine in zone A and the distance to the corresponding objects in zone B, assuming A and B are corresponding.

For breeding, we choose $\sqrt{n}$ best individuals and perform a uniform crossover on every possible pairing. The mutation operator replaces each position in an offspring genome with a valid random position. We discard all identical individuals and preserve best individual obtained so far (i.e. elitism of size 1). The algorithm stops after a given amount of time or if no better solution has been found in a number of past iterations.

Assuming we have the features placed, we can finally place the roads. We calculate a minimum spanning tree connecting all features and entrances on the zone and set the corresponding tiles as road tiles in this zone and all corresponding ones. The example is shown in Figure 4.

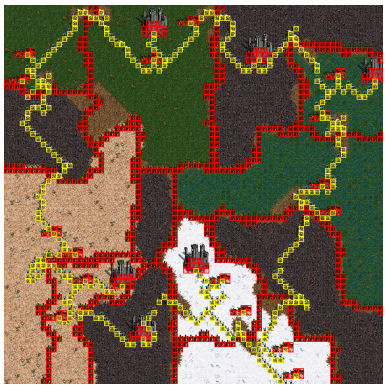


Fig. 4. The map after the placement of strategic features and roads (red tiles are zone borders, yellow tiles are roads)

F. Filling Space with Cellular Automata

It is characteristic for HoMM3 maps to contain irregularly placed obstacles, which allow placing treasures nearby and effectively guarding them. Since usually the exact locations of the obstacles do not heavily influence the map's properties, we can safely use cellular automata to fill the space randomly, and, if required, do some small fixes later.

In the standard case, cellular automata operate on two types of tiles: white and black. In each step, every cell can change its color according to the rules. In our case, we need to fill the interior of the zones without erasing borders separating zones or blocking roads set by the feature placing algorithm. Thus, we added two additional colors: *super-white* (which works as white but cannot be blacked) and *super-black* (which works conversely). This is more general then, e.g. overriding tiles with roads and borders after CA step because we are able to additionally parametrize automata behavior by giving separate weights to these special colors.

IV. Implementation

After all the steps presented in the previous section the main structure of the map is complete. The visualization of the example run we have described is shown in Figure 5.

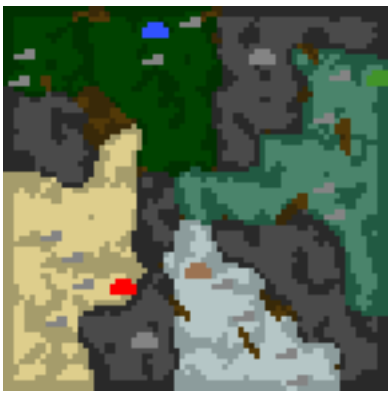


Fig. 5. The in-game minimap of the example map we have generated. This is an M-size map, for 4 players, and with all generation parameters set to default value.

Our map generator is written mainly in Lua, and partially in C++ and Python. We used Löve[7] for GUI. To generate maps in proper format, we used (and slightly fixed) C++ homm3tools library [34], for which we developed Lua API[8].

Graph embedding and visualization was done with NumPy[9], SciPy[10], and Matplotlib[11] Python packages. We used open source implementation of Sammon mapping from Github[12].

Evolutionary algorithm for features placement was run with population of size 100, mutation rate of 0.01, and time limit set to 1 second.

V. Discussion

We proposed a method that, in theory, can generate balanced map layouts for Heroes of Might and Magic III. The algorithm is highly modular, which proved to have both advantages and disadvantages. On one hand, we were able to independently develop and improve individual components. The randomized nature of some of them gives an opportunity to run them several times and pass the best result to the next step. It also means we can obtain varied results using the same parameter set. The partial visualizations of the process would not be possible without splitting the generator into separate phases.

On the other hand, ensuring truly balanced outputs turned out to be very hard. One of the biggest challenges comes from

[7]https://love2d.org
[8]https://github.com/radekmie/homm3lua
[9]http://www.numpy.org/
[10]https://www.scipy.org/
[11]https://matplotlib.org/
[12]https://github.com/tompollard/sammon

the cumulation of errors. Each step heavily depends on getting a reasonable input. If the previous phase fails to produce it, we usually have no way to fix it, since the components are almost completely independent. For example, MLML step does not know if the graph it is making can be nicely drawn on a plane. This sometimes results in scenarios, where Sammon mapping cannot provide a good embedding, and the final map is useless. Introducing some backtracking could help with this issue, but it is going to impossible to completely avoid the problem without some coupling between phases.

We also need to mention that at this point the balancing is only theoretical, and we did not have the opportunity to actually test the maps by playing them. The game HoMM3 is heavily based on moment-to-moment exploration, so getting the full experience requires all of the game-specific, low-level features to be present.

Although we developed our algorithm mainly for the purpose of generating Heroes of Might and Magic maps, it is generic in nature and can be adapted to other productions. The algorithm requires the existence of zones and strategic features, but these concepts occur commonly, and their equivalents can be easily defined in many strategy games.

If we consider real-time strategy game StarCraft [35], [36], the partitioning into zones is not so clear. However, we can still keep main routes as edges between some distinguished map areas and specify their terrain style (level of openness, high ground flag, the existence of chokepoints). We can identify resources (minerals, rich minerals, vespene gas) and controllable Xel'Naga Towers as strategic features. Destructible rocks could be encoded similarly to outers.

Another famous RTS, Warcraft III [37], is even more suitable for our method, mainly because of the number and significance of potential strategic features. This category includes not only gold mines, but also all neutral buildings (taverns, mercenary camps, marketplaces, etc.), creeps (neutral monsters of various strength that give experience and item rewards), and even teleports (called way gates).

As a slightly different example, we will mention Anno 1602 [38], an economic strategy game, usually taking place on a map consisting of multiple islands. Each island naturally maps to a zone, where resources, like gold or iron, are its features. Other strategic features may include island-specific crops, like cocoa and cotton, or the size of an island. The distinction between local and buffer zones is conventional in this case. We can simply assume that local zones form an archipelago of islands closest to the specific player, while buffer zones are islands equally distant from more than one player.

A. Future Work

The goal of a future work is to improve and extend the algorithm, and to finish the remaining parts of our HoMM3 map generator. In particular, we want to develop evaluation functions to estimate output quality after each stage of the procedure. It will let us to use generate-and-test approach, i.e. run each step several times and choose the best outcome.

We also plan to finish the full implementation of all HoMM3-specific features. Water and whirlpools should be implemented as the special type of buffer zones and teleport-like features. Underground map level should be formed by removing some buffer zones before mapping MLML onto a grid, and placing them below adjacent zones, so the subterranean gates can be placed in the overlapping areas. Grail should be placed in a buffer zone close to gameplay-based centers of the map, i.e. areas equally difficult to reach for all players.

We can also consider generating maps that are deliberately imbalanced. The simplest example for HoMM3 is a map containing AI-only players, who should have some handicaps (richer zones, more starting towns, etc.). This can be done by generating two different LML graphs – for human players and for AI-only players. Then, in the MLML phase, these graphs have to be merged in the right way. This is one of the non-trivial extensions we plan to implement and test in the future.

The remaining parts we need to include in our map generator are map aesthetics and low-level features placement. Apart from functional properties of the map, we should also take into account its visual aspects. HoMM3 contains various types of obstacles like trees, lakes, rocks, or mountains, ranging in size from 1×1 to 3×5 map tiles. To make generated map consistent and visually pleasant, the choice of obstacle should depend on its surroundings: terrain type, other obstacles, and strategic features. It is natural that sawmill should be near the trees and crystal cavern is placed in the mountains. Another example is a water wheel, which is a map object that should be placed on a river.

Zones and strategic features determine the outline of the game, but no less significant are low-level features scattered around the map: resources, artifacts, various special objects, and creatures guarding them. Their proper placement is the most important and challenging aspect of a future work, as the entire gameplay can be seen as a sequence of losses and gains. For example, a player loses some troops fighting wandering creatures and then picks up an artifact they were protecting.

In a less complicated domain, the solution could be to use evolutionary algorithms with balance-testing fitness function depending on AI agents simulating the players' behaviors [10], [39], [40]. In our case, to control the loss-gain loop on the map, we need to estimate players' capabilities based on our knowledge about the game mechanics.

It is for this reason that we introduced zone levels, as their semantics is closely correlated with player strength. Let us assume that we want a zone of level four to be attainable roughly on turn 15. To achieve that we need to estimate player's strength at that moment. It is possible knowing their starting town and the content of closer zones of lower levels. Thus, the remaining part is to place a proper creature at the entrance of this zone. We have developed simulation-based unit value estimation program for HoMM3[13], which is able to, for any amount and type of creature, estimate the number of other creatures needed to match its strength. We plan to

[13]https://github.com/maciek16180/h3-fight-sim

use these estimations to ensure the proper level of challenge when placing guarding creatures.

VI. Conclusions

In this paper, we focused on generating a balanced map layouts, with obstacles, partitioning into zones, and fair placement of the strategic features. We combined a variety of known methods including graph grammars, Voronoi diagrams, cellular automata, and evolutionary computation with novel approaches, like feature-redistribution graph grammar algorithm or MDS plus Bresenham-based layout solver.

The proposed algorithm was implemented in a map generator which we are developing for Heroes of Might and Magic III. Although it is not finished yet, it produces fully playable HoMM3 maps in proper `h3m` format. We argue that there is a community need for such a tool, especially because the recently released HD edition of the game does not contain the original map generator.

We presented a step-by-step description and visualization of our method, and discussed the details of its implementation. Lastly, we have shown that the presented algorithm is generic, and can be applied in other strategy games.

Acknowledgments

This work was supported in part by the National Science Centre, Poland, under the project number 2014/13/N/ST6/01817.

References

[1] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne, "Search-Based Procedural Content Generation: A Taxonomy and Survey," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 3, no. 3, pp. 172–186, 2011.

[2] Firaxis Games, *Civilization VI*, 2K Games, 2016.

[3] Blizzard Entertainment, *Diablo*, Blizzard Entertainment, 2012.

[4] D. Yu and A. Hull, *Spelunky*, Mossmouth, 2009.

[5] N. Shaker, J. Togelius, and M. Nelson, *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. Springer, 2016.

[6] J. Togelius, M. Preuss, and G. N. Yannakakis, "Towards Multiobjective Procedural Map Generation," in *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*, ser. PCGames '10, 2010, pp. 3:1–3:8.

[7] A. Liapis, H. P. Martínez, J. Togelius, and G. N. Yannakakis, "Adaptive game level creation through rank-based interactive evolution," in *2013 IEEE Conference on Computational Inteligence in Games (CIG)*, 2013, pp. 1–8.

[8] A. Liapis, G. N. Yannakakis, and J. Togelius, "Towards a Generic Method of Evaluating Game Levels," in *AIIDE*, 2014, pp. 30–36.

[9] P. T. Ølsted, B. Ma, and S. Risi, "Interactive evolution of levels for a competitive multiplayer FPS," in *2015 IEEE Congress on Evolutionary Computation (CEC)*, 2015, pp. 1527–1534.

[10] A. Liapis, C. Holmgård, G. N. Yannakakis, and J. Togelius, "Procedural Personas as Critics for Dungeon Generation," in *Applications of Evolutionary Computation*, ser. LNCS, 2015, vol. 9028, pp. 331–343.

[11] R. A. Agis, A. Cohen, and D. C. Martínez, "Argumentative AI Director Using Defeasible Logic Programming," in *Computer Games*, ser. CCIS, vol. 614, 2016, pp. 96–111.

[12] J. M. Font, R. Izquierdo, D. Manrique, and J. Togelius, "Constrained Level Generation Through Grammar-Based Evolutionary Algorithms," in *Applications of Evolutionary Computation*, ser. LNCS, vol. 9597, 2016, pp. 558–573.

[13] A. Liapis, "Piecemeal Evolution of a First Person Shooter Level," in *Applications of Evolutionary Computation*, ser. LNCS, 2018.

[14] D. Karavolos, A. Liapis, and G. N. Yannakakis, "Evolving Missions to Create Game Spaces," in *IEEE Conference on Computational Intelligence and Games (CIG)*, 2016, pp. 1–8.

[15] N. Shaker, M. Nicolau, G. N. Yannakakis, J. Togelius, and M. O'Neill, "Evolving levels for Super Mario Bros using grammatical evolution," in *2012 IEEE Conference on Computational Intelligence and Games (CIG)*, 2012, pp. 304–311.

[16] S. Dahlskog and J. Togelius, "Procedural Content Generation Using Patterns as Objectives," in *Applications of Evolutionary Computation*, ser. LNCS, vol. 8602, 2014, pp. 325–336.

[17] T. Mahlmann, J. Togelius, and G. N. Yannakakis, "Spicing up map generation," in *European Conference on the Applications of Evolutionary Computation*, 2012, pp. 224–233.

[18] M. Stephenson and J. Renz, "Generating varied, stable and solvable levels for angry birds style physics games," in *2017 IEEE Conference on Computational Intelligence and Games (CIG)*, 2017, pp. 288–295.

[19] L. Cardamone, G. N. Yannakakis, J. Togelius, and P. L. Lanzi, "Evolving Interesting Maps for a First Person Shooter," in *Applications of Evolutionary Computation*, ser. LNCS, vol. 6624, 2011, pp. 63–72.

[20] D. Loiacono and L. Arnaboldi, "Fight or flight: Evolving maps for cube 2 to foster a fleeing behavior," in *2017 IEEE Conference on Computational Intelligence and Games (CIG)*, 2017, pp. 199–206.

[21] J. Togelius, M. Preuss, N. Beume, S. Wessing, J. Hagelbäck, and G. N. Yannakakis, "Multiobjective exploration of the StarCraft map space," in *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games*, 2010, pp. 265–272.

[22] E. J. Hastings, R. K. Guha, and K. O. Stanley, "Evolving content in the Galactic Arms Race video game," in *IEEE Symposium on Computational Intelligence and Games*, 2009, pp. 241–248.

[23] S. Risi, J. Lehman, D. B. D'ambrosio, R. Hall, and K. O. Stanley, "Combining search-based procedural content generation and social gaming in the petalz video game," in *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2012, pp. 63–68.

[24] T. Soule, S. Heck, T. E. Haynes, N. Wood, and B. D. Robison, "Darwin's Demons: Does Evolution Improve the Game?" in *Applications of Evolutionary Computation*, ser. LNCS, vol. 10199, 2017, pp. 435–451.

[25] New World Computing, *Heroes of Might and Magic III*, The 3DO Company, 1999.

[26] G. N. Yannakakis and J. Togelius, *Artificial Intelligence and Games*. Springer, 2018.

[27] J. Rekers and A. Schurr, "A graph grammar approach to graphical parsing," in *Proceedings of Symposium on Visual Languages*, 1995, pp. 195–202.

[28] J. Dormans, "Adventures in level design: Generating missions and spaces for action adventure games," in *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*, ser. PCGames '10, 2010, pp. 1:1–1:8.

[29] K. Hartsook, A. Zook, S. Das, and M. O. Riedl, "Toward supporting stories with procedurally generated game worlds," in *Computational Intelligence and Games (CIG), 2011 IEEE Conference on*, 2011, pp. 297–304.

[30] S. Lee, A. Isaksen, C. Holmgård, and J. Togelius, "Predicting Resource Locations in Game Maps Using Deep Convolutional Neural Networks," in *AIIDE*, 2016, pp. 46–52.

[31] L. Johnson, G. Yannakakis, and J. Togelius, "Cellular Automata for Real-time Generation of Infinite Cave Levels," in *Workshop on Procedural Content Generation in Games*, ser. PCGames '10, 2010, pp. 10:1–10:4.

[32] J. W. Sammon, "A Nonlinear Mapping for Data Structure Analysis," *IEEE Trans. Comput.*, vol. 18, no. 5, pp. 401–409, 1969.

[33] E. R. Gansner, Y. Koren, and S. North, "Graph Drawing by Stress Majorization," in *Graph Drawing*, 2005, pp. 239–250.

[34] J. Åkerblom. (2015) homm3tools – Tools for Heroes of Might and Magic III. [Online]. Available: https://github.com/potmdehex/homm3tools

[35] Blizzard Entertainment, *StarCraft*, Blizzard Entertainment, 1998.

[36] ——, *StarCraft II: Wings of Liberty*, Blizzard Entertainment, 2010.

[37] ——, *Warcraft III: Reign of Chaos*, Blizzard Entertainment, 2002.

[38] Max Design, *Anno 1602*, Sunflowers Interactive, 1998.

[39] J. Kowalski and M. Szykuła, "Evolving Chesslike Games Using Relative Algorithm Performance Profiles," in *Applications of Evolutionary Computation*, ser. LNCS, 2016, vol. 9597, pp. 574–589.

[40] F. de Mesentier, S. Lee, J. Togelius, and A. Nealen, "AI as Evaluator: Search Driven Playtesting of Modern Board Games," in *AAAI 2017 Workshop on What's Next for AI in Games*, 2017.

Monte Carlo Methods for the Game Kingdomino

Magnus Gedda, Mikael Z. Lagerkvist, and Martin Butler
Tomologic AB
Stockholm, Sweden
firstname.lastname@tomologic.com

***Abstract*—*Kingdomino* is introduced as an interesting game for studying game playing: the game is multiplayer (4 independent players per game); it has a limited game depth (13 moves per player); and it has limited but not insignificant interaction among players. Several strategies based on locally greedy players, Monte Carlo Evaluation (MCE), and Monte Carlo Tree Search (MCTS) are presented with variants. We examine a variation of UCT called *progressive win bias* and a playout policy (*Player-greedy*) focused on selecting good moves for the player. A thorough evaluation is done showing how the strategies perform and how to choose parameters given specific time constraints. The evaluation shows that surprisingly MCE is stronger than MCTS for a game like Kingdomino. All experiments use a cloud-native design, with a game server in a Docker container, and agents communicating using a REST-style JSON protocol. This enables a multi-language approach to separating the game state, the strategy implementations, and the coordination layer.**

***Index Terms*—artificial intelligence, games, monte carlo, probabilistic computation, heuristics design**

I. Introduction

Implementations and heuristics for computer players in classical board games such as Chess, Go and Othello have been studied extensively in various contexts. These types of games are typically two-player, deterministic, zero sum, perfect information games. Historically, game theoretic approaches such as Minimax and similar variants such as Alpha-Beta pruning have been used for these kinds of games, dating back to Shannon in 1950 [1]. Recently more advanced techniques utilizing Monte Carlo methods [2] have become popular, many of them outperforming the classical game theoretic approaches [3], [4], [5].

The characteristics of the Monte Carlo-based methods also make them viable candidates for games with more complex characteristics such as multiplayer, nondeterministic elements, and hidden information [6]. With the recent emergence of more modern board games (also called *eurogames*), which often exhibit these characteristics, we naturally see more and more research successfully applying Monte Carlo-based methods to such games [7], [8], [9], [10].

Among the most common Monte Carlo-based methods we have Monte Carlo Evaluation (MCE) (also called *flat Monte Carlo*) [3] and Monte Carlo Tree Search (MCTS) [11], [12]. Flat Monte Carlo has shown some success [4] but is generally considered too slow for games with deep game trees [13]. MCTS has come to address the problems of MCE and become a popular strategy for modern board games. A plethora of enhancements have been presented for MCTS, both general and domain-dependent, increasing its performance even further for various games [14], [15], [16], [17], [18]. For shallow game trees it is still unclear which Monte Carlo method performs best since available recommendations only concern games with deep trees.

Kingdomino [19] is a new board game which won the prestigious *Spiel des Jahres* award 2017. Like many other eurogames it has a high branching factor but differs from the general eurogame with its shallow game tree (only 13 rounds). It has frequent elements of nondeterminism and differs from zero sum games in that the choices a player makes generally have limited effect on its opponents. The game state of each round can be quantified to get a good assessment of how well each player is doing which facilitates strong static evaluators. The difference in characteristics compared to previously examined eurogames can potentially render previous recommendations misleading.

We examine static evaluators, Monte Carlo Evaluation (MCE) and Monte Carlo Tree Search using the Upper Confidence Bound for Trees algorithm (UCT). Vanilla implementations of MCE and UCT are compared with various enhancements such as heuristics for more realistic playout simulations and an improvement to UCT which initially steers the selection towards more promising moves. All variants are thoroughly evaluated showing how to select good parameters.

The experimental focus is on heuristic design rather than building efficient competitive agents, i.e., the implementations are meant to be comparative rather than relying on low-level optimization tweaks. All agents are independent processes communicating with a game server using a JSON protocol.

II. Kingdomino

Kingdomino [19] is a modern board game for 2-4 players released in 2016 where the aim of each player is to expand a kingdom by consecutively placing *dominoes* provided in a semi-stochastic manner. A domino contains two *tiles*, each representing a *terrain* type and can have up to three *crowns* contributing to the score for its area. The goal is to place the dominoes in a 5x5 grid with large areas connecting terrains of the same type (using 4-connectivity) containing many crowns to score points.

A. Rules (3-4 Players)

You begin with your castle tile placed as the starting point of your kingdom and a meeple representing your king. In the first round, the same number of dominoes as there are kings in play are drawn from the *draw pile* and added to the *current draft*.

978-1-5386-4359-4/18/$31.00 ©2018 IEEE

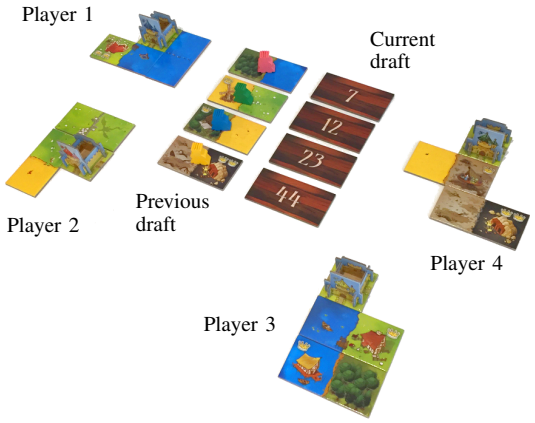


Fig. 1. Kingdomino in-game setup

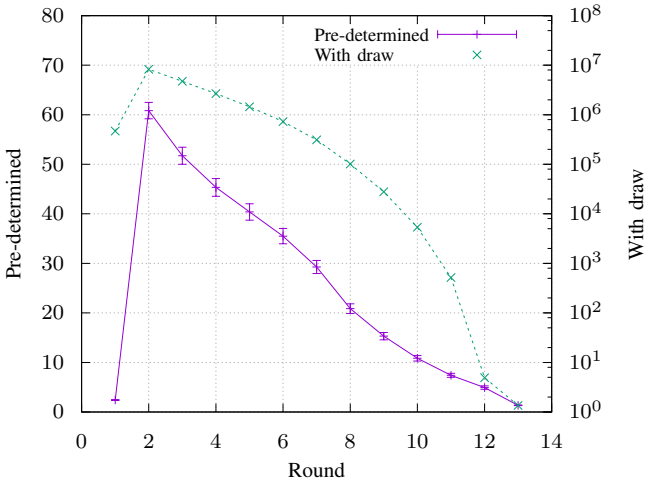


Fig. 2. Average branching factor per round for a random player when playing against three random opponents (1000 games). The error bars show the 95% confidence interval.

Each player then chooses one domino each from the current draft by placing their king on the chosen domino. When all dominoes in the draft have been chosen, the game moves on to the second round by drawing a new current draft from the draw pile. The previous current draft (the one that now has a king on each domino) becomes the *previous draft*.

In round two, and every consecutive round up until the last, the player with the king placed on the first domino in the previous draft adds the chosen domino to their territory, according to the connection rules, and chooses a new domino from the current draft by placing the king on the chosen domino. The other players then do the same placement-selection move in the order their kings are positioned in the previous draft. A placed domino must either connect to the castle tile or another domino matching at least one of its terrains (horizontally or vertically only). If you cannot add a domino to your kingdom, the domino will be discarded.

The last round works the same as the previous rounds with the exception that there are no more dominoes to draw from the draw pile and therefore there will be no current draft from which to choose any new dominoes.

The final score is the sum of the scores for each 4-connected area of the same terrain type. The score for each area is the number of tiles multiplied by the total number of crowns on the area. Note that for an area with no crowns, the score is zero. There are also two additional rules used in this paper (both part of the official game rules). The first is the *Middle Kingdom* rule, which states that you get an additional 10 points if your castle is in the center of the 5x5 grid. The second is the *Harmony* rule, which states that you get an additional 5 points if your territory is complete (i.e., no discarded dominoes).

For a complete description of the rules, including rules for 2 players, we refer to [19].

B. Game characteristics

Kingdomino is classified as a non-deterministic game since the dominoes are drawn randomly from the draw pile. All players have a similar goal and all players have complete information of the game state at all times, which means that it is also a symmetric perfect information game.

The number of possible draws from the deck is defined by the following formula. $\prod_{i=0}^{11} \binom{48-4i}{4} \approx 3.4 \cdot 10^{44}$ The most interesting thing about the number of possible draws is that it is significantly less than the total number of shuffles of the deck (around a factor of $3.6 \cdot 10^{16}$).

Fig. 2 shows the branching factor for each round. This is computed experimentally using 4-player games with the players choosing moves randomly (see Section V-A). Assuming that the branching factor for player p in round r is an independent stochastic variable B_{pr}, multiplying the expected value for the branching factor each round gives the expected value for the game tree size *given a predetermined deck shuffle*. Using the experimentally determined values for B_{pr}, the game tree size is approximately

$$E\left[\prod_{p=1}^{4}\prod_{r=1}^{13} B_{pr}\right] = \prod_{p=1}^{4}\prod_{r=1}^{13} E[B_{pr}] \approx 3.74 \cdot 10^{61}$$

When accounting for the number of possible draws from the deck, the number of Kingdomino games is around $1.27 \cdot 10^{106}$. This puts Kingdomino at a game tree complexity between Hex and Chess when accounting for all shuffles, and similar to Reversi/Othello for a pre-determined shuffle [20].

III. Strategies for Kingdomino

Agents can be implemented using a wide range of strategies. Here we focus on statistical evaluators such as Monte Carlo Evaluation and Monte Carlo Tree Search together with various enhancements. We also include some static evaluators to analyse game characteristics and use as reference agents when evaluating the statistical strategies.

A. Static Evaluators

Kingdomino generally has a solid score progression which makes it feasible to implement strong static evaluators by

computing the score of each player at every state of the game, unlike, e.g., the game of Go which has to rely heavily on statistical methods since domain-dependent move generators are very difficult to improve [4]. Also, considering Kingdomino is a perfect information game, any static evaluator with a greedy approach could potentially be competitive. We define two static evaluators, *Greedy Placement Random Draft* (GPRD) and *Full Greedy* (FG). GPRD places each domino in a greedy manner (to get maximum point increase) but selects dominoes randomly from the current draft while FG uses both greedy placement and selects greedily from the current draft. Both evaluators avoid moves that break the *Middle Kingdom* rule or result in single-tile holes. The FG evaluator is likely to act similar to an above average human player since it incorporates the visible domain knowledge to make realistic moves without using any search strategies.

B. Monte Carlo Methods

Monte Carlo methods such as Monte Carlo Evaluation (MCE) [3] and Monte Carlo Tree Search (MCTS) [11], [12] have recently been used successfully for building computer players in both classical two-player deterministic board games, such as Go [4], and more modern multiplayer non-deterministic board games, such as Settlers of Catan [7], Scotland Yard [8], and 7 Wonders [9].

1) Monte Carlo Evaluation: In flat Monte Carlo search (which we in this paper refer to as *Monte Carlo Evaluation*), each game state is represented by a node in a tree structure and the edges represent possible moves. The root node represents the current game state and its children represent the game states produced by each available move. The evaluation selects a child node randomly (using uniform sampling) and simulates a complete game from that node (referred to as a *playout*), using some *playout policy*, until termination. The selection-playout procedure is done repeatedly until either a maximum number of playouts have been reached or the time runs out. Each child node stores the average result from all its playouts, and the the max child is selected as the best move. Evaluators based on MCE have shown to be strong players in small classical games, such as 3x3 Tic-Tac-Toe, and play on par with standard evaluators on larger games [3].

The high exponential cost of searching trees with high branching factors makes global tree search impossible, especially under tight time constraints. However, the search depth of Kingdomino is shallow enough for MCE to potentially be a viable option since a shallow game tree facilitates high termination frequencies even at early stages in the game.

2) Monte Carlo Tree Search: Monte Carlo Tree Search expands on the functionality of Monte Carlo Evaluation by expanding the search tree asymmetrically in a best-first manner guided by statistics. A commonly used Monte Carlo Tree search algorithm for game play is UCT [11], which guides the search by computing the Upper Confidence Bound (UCB) for each node and select moves for which the UCB is maximal. The UCB is defined as

$$\mathrm{UCB} = \bar{X}_i + C\sqrt{\frac{\ln T}{T_i}}, \tag{1}$$

where $\bar{X}_i$ is the average payoff of move i, T is the number of times the parent of i has been visited, T_i is the number of times i has been sampled, and C is the exploration constant. For a full description of the UCT algorithm we refer to [11]. UCT, with enhancements such as domain-specific heuristics in the playout policies, has been shown to perform well for games with high branching factors [6].

C. Playout Policy Enhancements

The playout policy in its standard form uses random move selection throughout the playout. A common enhancement is to incorporate, potentially time expensive, domain-dependent heuristics to get more realistic playouts. We examine four different playout policies. The true random playout policy (TR) which chooses all moves randomly in the playout. The ϵ-greedy policy (ϵG) [6] which chooses moves randomly with ϵ probability and greedily with probability $(1 - \epsilon)$. The full greedy policy (FG) which chooses all moves greedily. And finally we use a playout policy we call the *player-greedy* policy (PG). It chooses the player's move greedily and all opponent moves randomly. Random opponent modelling has recently been applied successfully in multi-player tracks of General Video Game Playing (GVGP) AI competitions [21] but has, to our knowledge, not previously been applied to AI in board games. The player-greedy policy should be favourable in Kingdomino since the actions of the opponents generally have limited (but not insignificant) impact on the player. Its success in the GVGP setting can likely be attributed to the tight time constraints for opponent modelling in GVGP.

The ϵ-greedy and player-greedy strategies combine the advantage of domain knowledge with the speed provided by random move selection. With a low branching factor, there is a reasonable chance that good moves will be made with some frequency in random sampling. But games with large branching factors, such as Kingdomino, generally have many irrelevant, or even detrimental, moves. In these games the probability of playing out good moves during random playouts is relatively small, so there should be a large benefit to using informed simulation strategies.

D. Scoring Functions

The *scoring function* defines how the result of a playout is measured. The basic scoring function is the *Win Draw Loss* function (WDL) which simply gives a winning playout the score 1, a playout where the player is tied with an opponent for first place (a draw) the score 0.5, and a playout which is not a win or a draw the score 0. The reward model in Monte Carlo Evaluation facilitates more sophisticated scoring functions. One such function, which we refer to as the *Relative* scoring function (R), takes the player's score relative to the score of the highest scoring opponent $f = p_s/(p_s+q_s)$, where p_s is the player score and q_s is the opponent score. A third

third scoring function, which we refer to as the *Player* scoring function (P), simply uses the player's score. This function does not care whether the player wins or loses and only tries to maximize the player's own score.

E. MCTS Selection Enhancements

Among the popular enhancements for MCTS there are learning enhancements such as RAVE [16] and the history heuristic [14], [15]. They use offline information from previous games to guide the selection toward moves that have been successful in past games. Kingdomino has a low *n*-ply variance which means it could potentially benefit from learning enhancements [6]. However, in Kingdomino the reward of a single move is dependent on the game state, so the game state has to be incorporated in the offline information for each move. This has the effect of drastically decreasing the hit probability of a move while increasing lookup time.

A popular online enhancement is progressive bias [17] which guides the selection towards promising moves by using a – potentially time consuming – heuristic value which diminishes with increasing visits to the node. Here we use a selection enhancement which we call *progressive win bias* which combines progressive bias with a tweak that makes the heuristic value diminish with the number of node losses instead of the number of node visits. The tweak has successively been applied to the game Lines of Action [22] but has never been evaluated in a systematic fashion as presented here. We define progressive win bias as

$$W \frac{H_i}{T_i \left(1 - \bar{X}_i\right) + 1},$$

where H_i is the heuristic value, $\bar{X}_i$ is the average reward for the node, T_i is the number of node visits, and W is a positive constant which controls the impact of the bias. In this paper we use $H_i = S_i - S_{i-1}$ as heuristic, where S_γ is the player's score after move γ. The formula is simply added to the regular UCB in 1.

IV. Implementation

The implementation for the game is based on a server-client architecture. The server maintains all current, future, and past games, while a client agent can play in one or more games. A game is initiated with a set number of players, putting it in the list of future games. An agent can join a game, on which it receives a secret token enabling it to make moves for a player in the game. After enough players join the game, it is started. The game server has a graphical front-end showing all current and past games with full history for analysis and inspection.

Agents poll the server for the current game state: the kingdoms and their scores; the current and next draft; the current player; all possible moves; and all previously used dominoes. To make a move, the agent for the current player chooses one of the possible moves. The communication is based on a HTTP REST JSON API. The protocol gives enough information to enable stateless agents that only need remember their secret token. When joining a game, it is possible for an agent to register an HTTP callback endpoint that the server uses to notify the agent that its player is the current player.

The game server is implemented in Scala, and is packaged as a Docker container. This simplifies running the server in any setting, either on a remote server or locally. In particular, the choice of using standard web technologies for communication leads to a clean and simple separation of agents and the server.

At a one-day hackathon, 7 programmers could without preparation build rudimentary game playing agents in a variety of languages (Java, Scala, Python, Rust, and Haskell). The state representation and the full valid move list make it simple to implement static evaluators, without having to implement the full game logic. Naturally, for a more competitive client the full game logic needs to be implemented also in the client.

V. Experiments

Our experiments are intended to give insights into the game, to give guidance on what strategies and algorithms are useful, and how to tune parameters for the strategies. To compare strategies, we have made the choice to use static time limits per ply to study how well different strategies can make use of a specific time allotment without introducing the complexities of full time management.

Note that all games in these experiments are 4-player games (unless otherwise stated), so a when a strategy plays equally well as its opponent it will result in a 25% win rate. All intervals (in both figures and tables) represent the 95% confidence interval.

In board games the number of victories alone can be considered insufficient to determine the strength of a player. This is supported by the USOA (United States Othello Association) which uses the margin of victory as the single most important feature in determining a player's rating [3]. Therefore, most of our experiments use the victory margin to determine player strength.

A. Setup

All agents used in the experiments are written in Java and run on a single threaded 3.2 GHz Intel Core i7 with 12 GB RAM that is also running the game server. While the agents are not written to be the fastest possible, some care has been taken to keep the implementation reasonably fast. The goal is to facilitate comparison between the agents, not to implement a certain algorithm optimally.

B. Agents

We use three different static evaluator agents: the True Random (TR) agent, the Greedy Placement Random Draft (GPRD) agent, and the Full Greedy (FG) agent. The FG agent is used as reference player against which we evaluate all statistical players.

Each Monte Carlo Evaluation agent is implemented using flat Monte Carlo search and characterized by a playout policy/scoring function combination. We denote them by MCE-X/Y where X is the playout policy and Y is the scoring function.

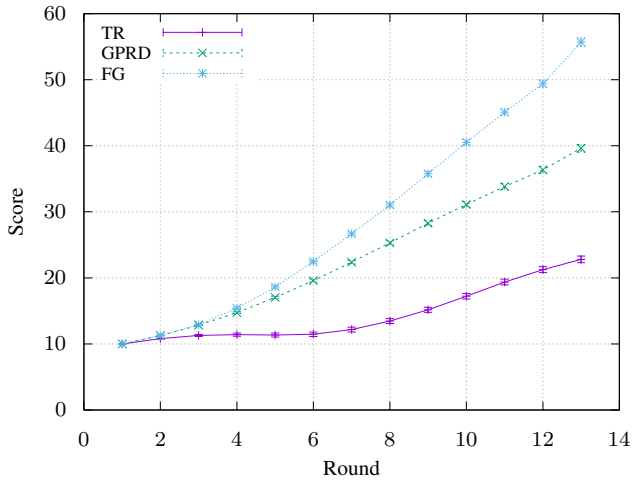


Fig. 3. Average scores against three TR opponents (1000 games).

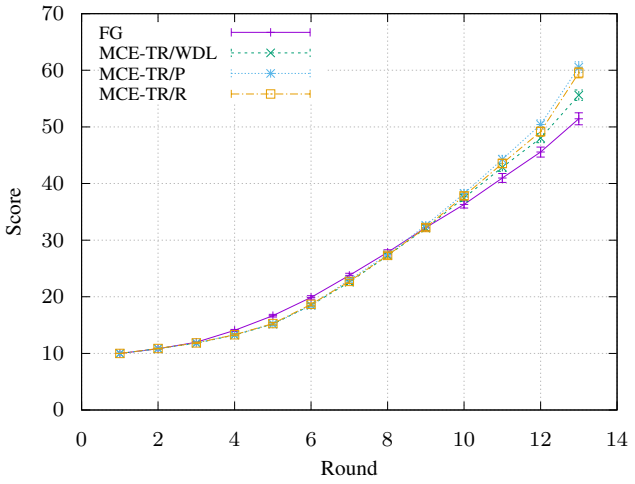


Fig. 4. Average scores against three FG opponents (500 games).

The Monte Carlo Tree Search agents all use the WDL scoring function and are therefore only characterized by playout policy and selection enhancements. The MCTS agents lack the possibility of using a relative scoring function but use maximum score increase as tie breaker for moves of equal win rate. We denote the standard MCTS agents by UCT-X, the MCTS agents using progressive bias by UCT_B-X, and progressive win bias by UCT_W-X, where X is the playout policy.

TABLE I
WIN PERCENTAGES FOR 1000 GAMES AGAINST THREE TR OPPONENTS.

Player Strategy	Opponent Strategy TR		
	Wins (%)	Draws (%)	Losses (%)
TR	223 (22.3)	29 (2.9)	748 (74.8)
GPRD	794 (79.4)	22 (2.2)	184 (18.4)
FG	977 (97.7)	2 (0.2)	21 (2.1)

C. Impact of Domain Knowledge

In the first experiment we wanted to quantify how basic domain knowledge affects strategies based on static evaluators. We did this by playing a True Random player (TR), a Greedy Placement Random Draft player (GPRD), and a Full Greedy player (FG) 1000 games each against three TR opponents and registered the number of wins, draws, and losses. We also registered the score after each round in every game to see the general score progression of each strategy.

The average score progression for the three different strategies over is shown in Fig. 3. All players start with 10p since the castle is within three tiles distance from the tile furthest away, thus fulfilling the Middle Kingdom rule. We can clearly see that the TR player had trouble increasing its score and even dipped around Round 5-6 due to breaking the Middle Kingdom rule. The GPRD player did a better job, showing that it is of great importance to select good positions for the placed domino. However, the score progression of the FG player indicates that it is of equal importance to also select a good domino from the current draft (the score for FG is approximately twice the score of GPRD when corrected for the scores of random moves).

The number of wins, losses, and draws for each strategy are shown in Table I. Here we see that the FG player truly outplayed the TR opponents, which was anticipated. More interesting is that the GPRD player only has approximately 79% win rate against the TR opponents. So while carefully selecting placements, making an uninformed selection from the current draft has a noticeable impact when played against random opponents.

D. Static vs Statistical Evaluators

In this experiment we investigated how simple statistical evaluation performs compared to the best static evaluator-based strategy. We also look at how different scoring functions affect the performance of the statistical evaluators. We did this by playing three Monte Carlo Evaluation players, each using a different scoring function and random selection playout policy, 500 games each against three FG opponents and compared the results to the same number of games played by a FG player against three FG opponents. The time limit was set to 5s per ply. The three Monte Carlo players were MCE-TR/WDL, which only counts the number of wins/draws/losses and chooses the move that maximises the number of wins, MCE-TR/P, which tries to maximise the player's final score, and MCE-TR/R, which tries to maximise the victory margin. The score progressions are shown in Fig. 4 and the final scores in Table II.

The experiment clearly shows that the statistical evaluators significantly outperform the FG player. It is interesting to see how the statistical evaluators select sub-greedy moves in the middle of the game to enable higher payoffs in the later parts of the game. It is also clear that MCE-TR/WDL does not reach as high final score as the other statistical evaluators. This is

TABLE II
AVERAGE SCORES FOR 500 GAMES AGAINST THREE FG OPPONENTS.

Player Strategy	Avg. Score
FG	51.4 (2.1)
MCE-TR/WDL	55.6 (1.8)
MCE-TR/P	60.6 (1.9)
MCE-TR/R	59.5 (1.8)

most likely a result of the WDL scoring function's lack of score information which renders it incapable of discriminating between branches where all leaf nodes result in a win while it is in the lead. Since each node only stores the winning average, it will not be able to determine which branch will lead to a higher final score. Also, the R and P scoring functions are more robust against the recurring stochastic events. There is no significant difference in performance between the Player scoring function and Relative scoring function.

E. Enhanced Playout Policies

In this experiment we investigated the effect of different enhancements to Monte Carlo Evaluation by incorporating domain knowledge into the playout policies. We did this by playing Monte Carlo Evaluation players, both with and without domain knowledge, against three FG opponents and compared the results. The players we used were MCE-TR/R, which has no domain knowledge at all and only selects moves randomly for both the player and opponents in the playouts, MCE-ϵG/R with $\epsilon = 0.75$, which uses random selection in 75% of the times in the playout and greedy selection 25% of the times, MCE-PG/R, which uses greedy selection for the player and random selection for the opponents in the playouts, and MCE-FG/R, which uses greedy selection for all moves in the playouts. We used the relative scoring function since its goal aligns with the measure of player strength and facilitates easier analysis of the result plots.

Since all games in the experiment were 4-player games and ϵ was set so that greedy selection will be used 25% of the time, the number of greedy move evaluations would be the same for both MCE-ϵG/R and MCE-PG/R and should result in approximately the same playout frequency for the two simulation strategies. This will tell us how important accurate opponent modelling is in Kingdomino.

Fig. 5 shows the victory margin under various time constraints for the different strategies (each point represents 200 games). In addition to the Monte Carlo Evaluation game strategies, the result from playing 200 games with an FG player against three FG opponents is also shown (the solid red line with the 95% confidence interval as dotted red lines). Fig. 6 shows the number of playouts per second for each playout policy.

The experiment shows that the FG evaluator is competitive to the statistical evaluators under tight time constraints. It is comparable to MCE-TR/R, and outperforms all the others, when the time is capped to $0.1s$ per move. It also shows

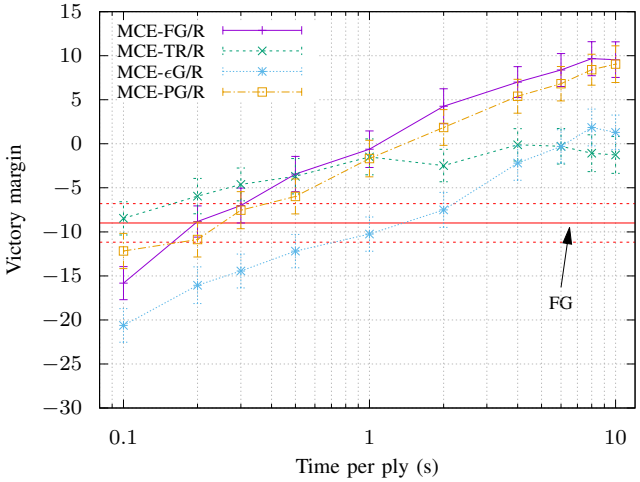


Fig. 5. Average victory margins against three FG opponents.

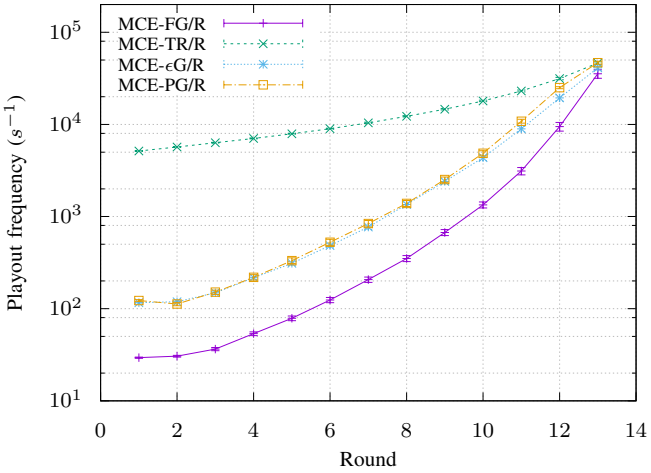


Fig. 6. Average playout frequency (200 games).

that the best knowledge-based statistical evaluators need approximately $0.5 - 1s$ time per move for the extra heuristic computations to pay off compared to selecting playout moves randomly, but they consistently outperform the random playout policy for move times $> 1s$. It also shows that it is more important to model the player's own move realistically than the moves of the opponent. This is clear from the difference in performance between MCE-PG/R and MCE-ϵG/R when having approximately the same playout frequencies. Furthermore, if we compare MCE-PG/R to MCE-FG/R we see that realistic opponent modelling is disadvantageous for short ply times ($< 0.2s$). This is natural since realistic opponent modelling is costly and MCE-FG/R will only have time for few playouts before selecting its move, while MCE-PG/R can produce more playouts and have a better statistical sample when choosing its move. However, once the number of playouts go up ($> 0.1s$) we see that realistic opponent modelling consistently outperforms the player-greedy strategy, although not by much.

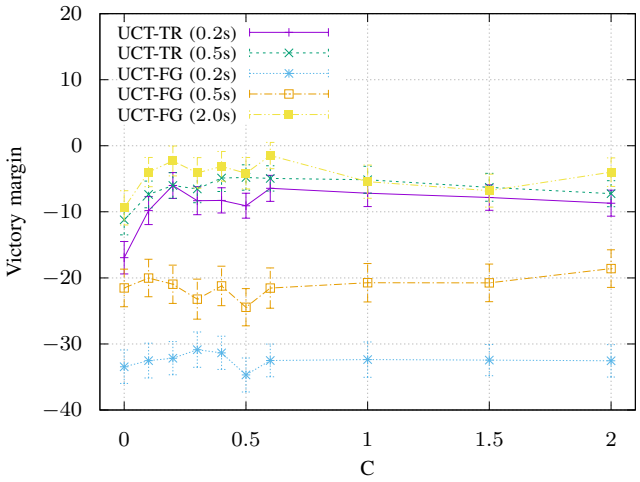


Fig. 7. Average victory margins against three FG opponents.

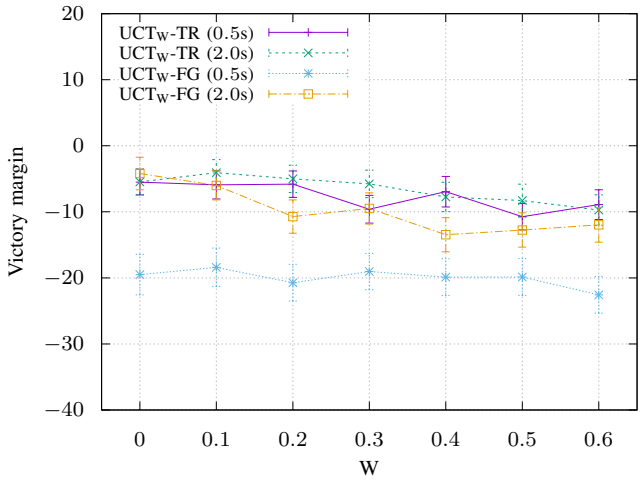


Fig. 8. Average victory margins against three FG opponents.

F. Tree Search

We examined the UCB exploration constant C by playing an UCT-TR and an UCT-FG player against three FG players for various values of C. The result is shown in Fig. 7. The experiment shows that $C = 0.6$ is a suitable value for players with many playouts per ply and $C \geq 1.0$ for strategies with few playouts per ply. A theory is that due to Kingdomino's frequent stochastic events, a move requires numerous playouts to accumulate a representative reward. So there is a risk of focusing the tree expansion on high-reward moves before all moves get representative rewards. Therefore, players with few playouts per ply should perform better with a higher exploration constant.

We also examined the impact constant W for progressive bias and progressive win bias by playing a UCT_W-TR player and a UCT_W-FG player, both with $C = 0.6$, against three FG opponents for various values of W. The result is shown in Fig. 8. It shows that we get the highest performance impact for $W = 0.1 \sim 0.2$ and after that the performance decreases with W.

G. Comparing Strategies

Table III shows the performance of all strategies for 200 games played against three FG opponents. The 95% confidence intervals are in the range $[3.5, 6.0]$ for all entries, with the majority near the lower limit. The highest performer for each time constraint is marked by a dark blue box. Performances within 5% (10%) of the best are marked by a light (lighter) blue box. The UCB exploration constant was set to $C = 0.6$ for all UCT strategies and the the bias impact factor was set to $W = 0.1$ for UCT_B-* and UCT_W-*.

The results show that for each time constraint the best MCE variant consistently outperforms all variants of UCT. A possible theory is that UCT is hampered by its WDL scoring function, but further experiments verifying this hypothesis is outside the scope of this paper. The true random playout policy variant (MCE-TR/R) excels for short ply times $t < 0.5s$. After that the full greedy playout policy variant (MCE-FG/R) gets enough time each ply to produce rewards representative enough to reliably select trajectories in the game tree that outperform the the random playout policy, in spite of the significantly higher playout frequency of the random playout policy. The MCE-PG performs almost on par with MCE-FG which indicates that allocating time for accurate opponent modelling only has a small gain compared to using random move selection for the opponents.

The results also show that the UCT enhancements improve the results for tight time constraints ($t < 0.2s$), which is expected due to few playouts, but are otherwise on par with regular UCT.

VI. Conclusions and Future Work

This paper introduces Kingdomino as an interesting game to study for game playing. The shallow game tree and relatively limited interaction between players of Kingdomino combined with the stochastic nature and possibility to evaluate partial game states is particularly interesting. The results indicate that for games such as Kingdomino, MCE is superior to UCT, which would infer new recommendations on the suitability of MCE for games of similar complexity. This is especially interesting, given that an MCE evaluator is significantly easier to implement correctly and efficiently than full UCT.

The *player-greedy* playout policy is surprisingly effective, balancing exploration power with (expensive) local evaluation. Our belief is that this is due to the limited (but not insignificant) interaction among players in Kingdomino, but further experiments in other games are needed to verify this hypothesis. The *progressive win bias* selection improvement shows promise as a way to combine a heuristic evaluation with the current knowledge gained from the exploration, but further experiments in other settings better suited for the UCT is needed to analyse its impact.

Our evaluation uses thorough systematic examination of all constants involved to avoid the presence of magic numbers

TABLE III
AVERAGE VICTORY MARGINS FOR 200 GAMES AGAINST THREE FG OPPONENTS.

Strategy	Time per ply									
	0.1s	0.2s	0.3s	0.5s	1.0s	2.0s	4.0s	6.0s	8.0s	10.0s
FG	-9.0	-9.0	-9.0	-9.0	-9.0	-9.0	-9.0	-9.0	-9.0	-9.0
MCE-TR/R	-8.5	-5.9	-4.6	-3.7	-1.5	-2.5	-0.1	-0.3	-1.1	-1.3
MCE-FG/R	-15.8	-8.8	-7.0	-3.4	-0.6	4.3	7.0	8.4	9.7	9.6
MCE-PG/R	-12.2	-10.9	-7.5	-6.0	-1.7	1.9	5.4	6.8	8.4	9.0
MCE-ϵG/R	-20.6	-16.0	-14.5	-12.2	-10.3	-7.5	-2.2	-0.3	1.9	1.3
UCT-TR	-13.5	-6.4	-7.3	-4.9	-5.5	-4.4	-3.5	-5.2	-7.2	-4.4
UCT-FG	-38.3	-32.5	-29.4	-21.6	-12.0	-1.5	-0.2	3.5	4.0	3.9
UCT-PG	-25.8	-20.7	-15.5	-15.3	-13.9	-10.4	-7.1	-7.4	-6.2	-4.1
UCT-ϵG	-33.3	-24.0	-16.2	-15.7	-9.0	-7.0	-3.1	-4.0	-2.6	-1.3
UCT_B-TR	-10.1	-7.4	-6.1	-7.9	-4.6	-6.7	-4.8	-4.3	-2.9	-4.5
UCT_B-FG	-39.8	-30.6	-29.7	-21.6	-11.7	-5.9	-0.1	3.2	3.2	1.4
UCT_B-PG	-25.5	-20.8	-19.7	-15.4	-13.4	-9.1	-7.6	-6.3	-4.5	-12.4
UCT_B-ϵG	-31.5	-25.2	-20.2	-16.3	-10.7	-8.1	-4.1	-2.6	-1.9	-2.9
UCT_W-TR	-11.4	-6.7	-7.3	-5.9	-4.6	-4.1	-5.8	-5.0	-4.0	-4.5
UCT_W-FG	-42.6	-33.0	-30.3	-18.4	-13.9	-6.0	-2.5	0.6	1.2	1.4
UCT_W-PG	-29.2	-24.3	-20.0	-19.7	-15.4	-16.9	-13.1	-12.2	-13.9	-12.4
UCT_W-ϵG	-30.5	-23.0	-22.8	-16.6	-13.2	-6.6	-5.4	-3.1	-2.7	-2.9

which frequently occur without explanation in many similar papers in the field. It also uses new and illuminating graphs for showing the impact of different choices. In particular, the usage of victory margin in favour of win percentages is very powerful for a multi player score maximization game such as Kingdomino. These graphs have helped us gain new insights into both the game and how our strategies perform.

For future work one MCTS enhancement alternative could be a learning heuristic that keep offline information on the success of placement positions for different kingdom patterns. Experienced human players tend to place dominos in a structured pattern to avoid single tile holes in the kingdom. It would also be interesting to implement agents using completely different strategies such as deep reinforcement learning.

The code for the Kingdomino game server can be downloaded from `https://github.com/mratin/kdom-ai`, and the AI implementations can be downloaded from `https://github.com/mgedda/kdom-ai`.

ACKNOWLEDGEMENTS

We thank all participants at Tomologic who implemented agents and discussed strategies with us.

REFERENCES

[1] Shannon, C.E.: XXII. Programming a computer for playing chess. The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science **41**(314) (1950) 256–275
[2] Metropolis, N., Ulam, S.: The monte carlo method. Journal of the American statistical association **44**(247) (1949) 335–341
[3] Abramson, B.: Expected-outcome: A general model of static evaluation. IEEE TPAMI **12**(2) (1990) 182–193
[4] Bouzy, B., Helmstetter, B.: Monte-carlo Go developments. In: Advances in computer games. Springer (2004) 159–174
[5] Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., *et al.*: Mastering the game of Go with deep neural networks and tree search. Nature **529**(7587) (2016) 484–489
[6] Sturtevant, N.R.: An analysis of UCT in multi-player games. In: Intern. Conference on Computers and Games, Springer (2008) 37–49
[7] Szita, I., Chaslot, G., Spronck, P.: Monte-carlo tree search in Settlers of Catan. In: Advances in Computer Games, Springer (2009) 21–32
[8] Nijssen, J.P.A.M., Winands, M.H.M.: Monte carlo tree search for the hide-and-seek game Scotland Yard. IEEE Transactions on Computational Intelligence and AI in Games **4**(4) (2012) 282–294
[9] Robilliard, D., Fonlupt, C., Teytaud, F.: Monte-carlo tree search for the game of "7 Wonders". In: Workshop on Computer Games, Springer (2014) 64–77
[10] Browne, C.B., Powley, E., Whitehouse, D., Lucas, S.M., Cowling, P.I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., Colton, S.: A survey of monte carlo tree search methods. IEEE Transactions on Computational Intelligence and AI in games **4**(1) (2012) 1–43
[11] Kocsis, L., Szepesvári, C.: Bandit based monte-carlo planning. In: European conference on machine learning, Springer (2006) 282–293
[12] Coulom, R.: Efficient selectivity and backup operators in monte-carlo tree search. In: International conference on computers and games, Springer (2006) 72–83
[13] Nijssen, J.P.A.M.: Monte-Carlo Tree Search for Multi-Player Games. PhD thesis, Maastricht University, The Netherlands (12 2013)
[14] Schaeffer, J.: The history heuristic. ICCA Journal **6**(3) (1983) 16–19
[15] Winands, M.H.M., van der Werf, E.C.D., van den Herik, H.J., Uiterwijk, J.W.H.M.: The relative history heuristic. In: International Conference on Computers and Games, Springer (2004) 262–272
[16] Gelly, S., Silver, D.: Combining online and offline knowledge in UCT. In: Proceedings of the 24th international conference on Machine learning, ACM (2007) 273–280
[17] Chaslot, G.M.J.B., Winands, M.H.M., Herik, H.J.v.d., Uiterwijk, J.W.H.M., Bouzy, B.: Progressive strategies for monte-carlo tree search. New Mathematics and Natural Computation **4**(03) (2008) 343–357
[18] Nijssen, J.P.A.M., Winands, M.H.M.: Enhancements for multi-player monte-carlo tree search. In: International Conference on Computers and Games, Springer (2010) 238–249
[19] Cathala, B., Bouquet, C.: Kingdomino (2016)
[20] Wikipedia contributors: Game complexity — Wikipedia, the free encyclopedia (2018) [Online; accessed 2018-02-15].
[21] Gaina, R.D., Couëtoux, A., Soemers, D.J.N.J., Winands, M.H.M., Vodopivec, T., Kirchgeßner, F., Liu, J., Lucas, S.M., Perez-Liebana, D.: The 2016 two-player gvgai competition. IEEE Transactions on Computational Intelligence and AI in Games (2017)
[22] Winands, M.H.M., Bjornsson, Y., Saito, J.T.: Monte carlo tree search in lines of action. IEEE Transactions on Computational Intelligence and AI in Games **2**(4) (2010) 239–250

Evolving Number Sentence Morphing Puzzles

Daniel Ashlock
Department of Mathematics and Statistics
University of Guelph
Guelph, Ontario, Canada
dashlock@uoguelph.ca

Courtney Kolthof
Department of Mathematics and Statistics
University of Guelph
Guelph, Ontario, Canada
ckolthof@uoguelph.ca

***Abstract*—Number sentence morphing is an example of a type of edit puzzle. Edit puzzles choose a class of objects and a collection of editing operators. The goal of the puzzle is to change an initial configuration into a final one, using the edit operations, with the intermediate objects remaining within the selected class of of objects. Changing one English word into another by changing single letters, with all the intermediate steps also consisting of English words, is a typical example of an edit game. A number sentence is a well formed mathematical object, which can be used to generate a type of edit puzzle called Number sentence morphing. This puzzle has the player transform a false number sentence into a true one by removing and replacing symbols from a number sentence while leaving the sentence well formed. This type of puzzle is intended for teaching students the notion of well-formation in a game setting as well as providing practice with basic arithmetic. An evolutionary algorithm is used to construct number sentence morphing puzzles while certifying that they can be solved and requiring a minimal number of steps in a solution. The search for number sentences is template driven so particular forms of sentences are searched for in their own sets of runs. An unexpected outcome of this study is that the search landscapes for different templates are very different.**

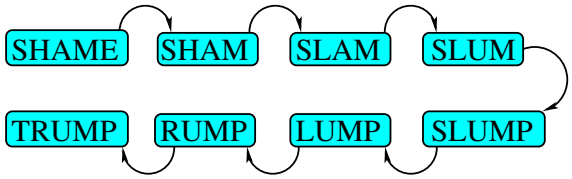
***Index Terms:* edit puzzles, procedural content generation, educational content generation, evolutionary algorithms**

I. Introduction

This publication investigates a type of automatic content generation in which we create instances of a puzzle, *number sentence morphing*, with an evolutionary algorithm. Number sentence morphing is a type of *edit game*. Edit games are games where a collection of objects and rules for editing those objects are chosen. The player then attempts to use successive edits to transform a given object into a target object by applying the editing rules. Suppose we may change, add, or delete single letters from a word. Then Example 1 shows an edit path between two words in which all intermediates are also English words.

Example 1: If the set of objects we are working with are English words and the edit operations are to add, delete, or change a single letter then the following sequence demonstrates the transformation of one work into another through a chain of intermediate strings that are also English words.

The authors thank the University of Guelph and Canadian Natural Science and Engineering Research Council of Canada (NSERC) for supporting this work.



Example 1 is a *word morphing* puzzle. For this type of puzzle the player is given a starting word and a goal word and asked to transform the first into the second by adding, deleting, or substituting letters with the added constraint that each intermediate step must itself be a word. A word morph is scored by the number of editing moves used with the goal being to find the smallest number of edits required. The example used seven moves – but someone with a good vocabulary might still manage to get there in fewer. A teacher might also insist that the intermediate words be polite, or come from the current vocabulary list for the class. This latter case makes creating such puzzles more challenging because of the highly restricted set of words available. In the case of the puzzle in Example 1, style points are awarded for the appropriateness of the intermediate words to the situation suggested by the choice of initial and target word.

This study adapts word morphing to a type of puzzle intended to exercise students on basic arithmetic and at least one foundational concept, *well-formation*. A *number sentence* is a string of digits and arithmetic operations that includes one equals sign. In general a construct is *well formed* if it obeys a formal grammar. A number sentence is well formed if numbers and operations alternate with one another and if the expressions on either side of the equals sign begin and end with numbers. A number sentence can be false or true as well, depending on the values on either side of the equals sign. Example 2 give examples of number sentences that are not well formed and of both true and false number sentences that are well formed.

Example 2:

23+=16-7	Not well formed
23=16-7	Well formed but false
23=16+7	Well formed and true.

A *number sentence morphing puzzle* (NSM puzzle) is an edit puzzle on the space of well-formed number sentences. Number sentence morphing was introduced in [4]. The edit operations consist of moving one symbol from the number sentence to a *symbol pool* or re-inserting one of the symbols

978-1-5386-4359-4/18/$31.00 ©2018 IEEE

from the symbol pool into the number sentence. The goal is to make a false well-formed number sentence into a true well-formed number sentence. These puzzles are intended to introduce the notion of well formation to elementary students, as well as letting them practice arithmetic, in a game setting. Example 3 gives an example of the solution of a NSM puzzle.

Example 3: This puzzle starts with $7 * 2 - 9 = 1$ and an empty symbol pool.

Move	Sentence	Pool
Start	7*2-9=1	
1	72-9=1	{*}
2	2-9=1	{*,7}
3	29=1	{-,*,7}
4	9=1	{-,*,2,7}
5	97=1	{-,*,2}
6	9-7=1	{*,2}
7	9-7=12	{*}
8	9-7=1*2	

The puzzle in this example was evolved to be solvable in eight steps, but no smaller number of steps, with the added condition that all the symbols must be used in the final true number sentence.

The puzzles created in this study are intended to be presented to elementary education teachers as a way of enabling practice of basic arithmetic operation and introducing the philosophical concept of a well formed formula in a form that is both engaging and instructive. Rules for a competitive game based on number sentence morphing are given in [4]. The puzzle can also be used as a solitaire with the goal of solving the problem in the fewest possible steps.

II. Background

Procedural content generation [9] is the use of procedural methods to generate content for games [3], the web [12], terrain maps [8], [5], or any of a number of other application domains [10]. The technique used in this study is an example of *search based* procedural content generation. A survey of the early years of this technique appears in [14]. Other examples include [11], [2].

The use of procedural content generation to create educational materials is a relatively new application. In [13] the authors make the case that digital content generation is a good way to generate a sufficiently large collection of examples that students are more likely, through exposure to the rich collection of examples, to inductively learn the underlying principles.

There are many sorts of edit puzzles and design must look closely at not only the class of objects but the editing rules. The editing rules in Example 1, for example, permit the addition and deletion of letters. A more usual version of word morphing permits only substitutions, making the number of paths between pairs of words much smaller and so the puzzles more difficult. Try to solve the puzzle in Example 1 with only substitutions: this is harder, but possible.

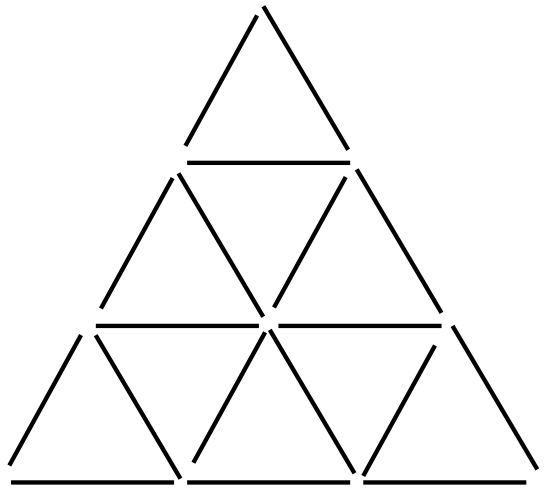


Fig. 1. A "remove toothpicks" puzzle starting configuration.

Another type of edit puzzle is one using physical manipulatives. Figure 1 shows and example of the starting configuration of a puzzle done with toothpicks. There are thirteen triangles (of three different sizes) formed by the toothpicks. A puzzle might ask that the smallest number of toothpicks be removed to generate a puzzle with five, six, or seven triangles.

Again, two things are needed to define a class of edit puzzles. The first is the collection of configurations that are allowed, the second is a set of editing operations. In the word morphing example, only English words are allowed, making the puzzle both more difficult than if any arbitrary string of letters is permitted, and also rendering the puzzle a test of the player's vocabulary. For the toothpick puzzle the configurations were anything that you could get from the original configuration by removing toothpicks. The edit moves consisted of the ways to remove one toothpick. Edit games can be thought of as being contained in a single, overarching mathematical structure: *graph theory*.

An article on the use of graphs in game and puzzle design is available in [1] while the graph algorithm used for solving or designing edit puzzles is explained in [6]. Both these articles appear in earlier editions of *Game and Puzzle Design*. Briefly, a graph G is a collection of objects or *vertices*, denoted $V(G)$ or just V and a set of connections between pairs of objects called *edges*, denoted $E(G)$ or E. For an edit game, the vertices are the permitted configurations and the edges are pairs of configurations such that one can be transformed into another with an editing move. This graph is the object searched during fitness evaluation. A part of one of the graphs, demonstrating three distinct solutions to a NSM puzzle, is shown in Figure 2.

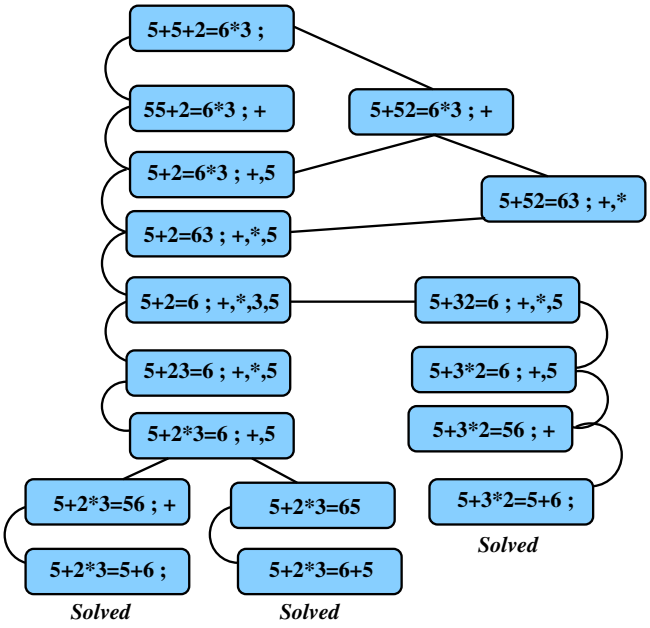


Fig. 2. A part of the graph that arises from the edit structure of number sentence morphing.

III. Design of Experiments

The design of number sentence morphing given in [4] permitted the use of parenthesis which are not used in this evolutionary study. The fitness function is a search of the space of legal moves and the use of parenthesis explodes the size of the search tree. This search is performed with dynamic programming [6]. The authors intend to restore parenthesis after a more subtle search strategy for the space of trees has been found; in the final section the case is also made that parenthesis can simply be added to puzzles constructed without them.

A. The Evolutionary Algorithm

The evolutionary algorithm used is a simple one, but with some unusual properties. Each run of the algorithm uses a template for a well-formed number sentence, e.g. **dodod=dd**. When creating population members, the letter **d** is replaced with a digit and the letter **o** is replaced with an operation. This permits the generation of an initial population of number sentences that all agree on their form.

The algorithm uses uniform crossover with a 25% probability of exchanging symbols at each position. The values 10%, 25% and 50% were compared in preliminary experimentation and the parameter is soft, in part because the strings being evolved are quite short. Use of the template ensures that crossover preserves well formation. Mutation replaces from 1 to some maximum permitted number of symbols, based on the *maximum number of mutations* (MNM) parameter. A character is replaced with a distinct character of the same type (digit or operation).

For fitness, three qualities are evaluated for a number sentence. The researcher states a minimum difference between the two sides of the equals sign that is acceptable, usually zero, a minimum number of steps that the problem should require for solution, and an acceptable number of remaining symbols in the symbol pool, again usually zero. Any sentence that can be edited to achieve the desired minimum difference is more fit than one that cannot. Any sentence that requires more steps to solve is more fit than another, although the algorithm does not search beyond a stated minimum desired number of steps, and, finally, a number sentence is more fit if it is closer to the desired remaining number of symbols in the pool in its best solution meeting the other criteria. These three fitness criteria are applied serially with the less important ones serving as tie-breakers for the more important ones. This type of composite fitness function is called a *lexical* fitness function [7].

The three qualities, solubility, solution length, and remaining symbol pool size are evaluated by performing a breadth first search of the tree of legal moves, to a depth equal to the desired minimum number of solution steps. This is a fairly expensive fitness function, as it examines a combinatorial space of possible morphs of the number sentence under test. The algorithm stops when it finds a formula that meets the desired goal for all three of the evaluated qualities.

IV. Experiments performed

Initial runs performed while testing the NSM evolution software found that the problem is almost trivial (solutions appear in the original population) for smaller requested minimum solution lengths. The search problem grows in difficulty as the desired minimum solution length is increased. To better understand this phenomena, a sampling study for the templates **dod=dod** and **dodod=dod** was performed, extracting the minimum steps to solve 4000 randomly generated number sentences for each template.

A series of experiments were done to try to set the MNM parameter. Five sets of 100 runs were performed on the template **ddod=ddodd** with $MNM \in \{1, 2, 3, 4, 5\}$ and four sets of runs were performed on the template **dodod=dod** with $MNM \in \{1, 2, 3, 4, 5\}$. The former runs used an initial population size of 1000 while the latter used an initial population size of 40. All of these experiments used a desired minimum number of steps to solution of eight. Shorter required minimum solution steps yield problems that can be solved by random sampling.

A barrier to performing larger numbers of experiments is the speed of the software. The fitness function must enumerate a very large space of morphs for each formula. Hand examination showed that it was usually possible to solve a NSM puzzle with no more than one arithmetic symbol (+, -, *) in the symbol pool; this restriction was imposed by saying that an arithmetic symbol could only be the first symbol added to an empty symbol pool. This run used initial population size 40 and MNM 5.

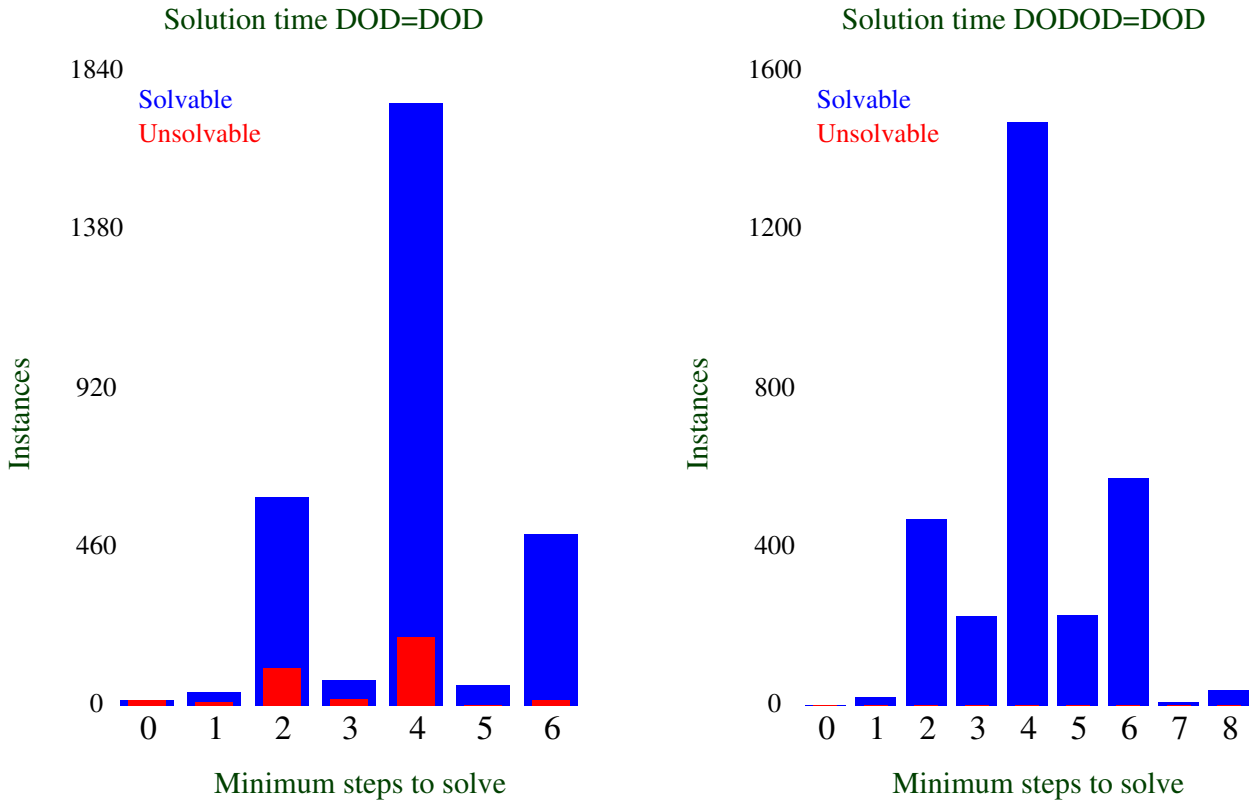


Fig. 3. Distribution of the number of steps needed to find the best possible result for 4000 number sentences of the sort used in the initial population of the evolutionary algorithm. Blue denotes sentences that can be solved to zero difference; red those that cannot. Two templates, given above, were sampled.

An additional collection of 100 runs were performed for population size 100 and $MNM = 1$ using the templates **ddod=dodd** and **dodod=d** simply to test additional templates and acquire a richer collection of puzzles.

V. Results and Discussion

The results of the sampling study are shown in Figure 3. These demonstrated that a majority of randomly generated number sentences are solvable with zero minimum difference. Figure 3 shows that most number sentences for the simpler template could be solved and almost all (all but five) of the randomly generated number sentences for the template **dodod=dod** solved with zero difference.

The thing that makes it difficult to find NSM puzzles that have relatively long minimum number of steps to solution is that a vast majority of the number sentences can be brought to their best possible minimum difference in four steps. Not one of the 4000 **dod=dod** templates required more than six steps. This means that the hard work for the evolutionary algorithm is finding puzzles with a large minimum number of steps to solve.

Figure 4 shows the result of the mutational studies. These runs demonstrate that mutation is a fairly soft parameter. Using a larger population size, which is quite expensive, yields a substantial improvement in the failure rate. Combining failure rate and number of mutation information yields a mild preference for $MNM = 1$ on **ddod=ddodd** and no preference on **dodod=dod**.

The added restriction that only one arithmetic symbol could be in the number sentence morphing pool yielded a huge advantage in both time to solution and the error rate. The effect of this restriction is to reduce the size of the search tree needed to find formulas. This experiment suggests that restricting the search tree for fitness evaluation is the clear direction for improving the speed of the system.

The drop in the error rate for the restricted run – from 31 to 0 given the common $MNM = 5$ parameter – is very interesting. When the fitness function of an evolutionary algorithm is changed, the fitness landscape where search takes place also changes. Increased speed should permit the researcher to drop the failure rate by increasing the number of fitness evaluation: that is not what happened. The same number of fitness evaluations, 40 in the initial population and 1000 during evolutionary search, were used in both collections of runs.

This means that the fitness landscape of the new fitness function is more evolution-friendly for reasons that are not obvious. There are a number of ways to prune or restrict the search tree that generates morphs. These need to be evaluated not only for their speed increase but on the degree to which they make the problem more or less evolution-friendly.

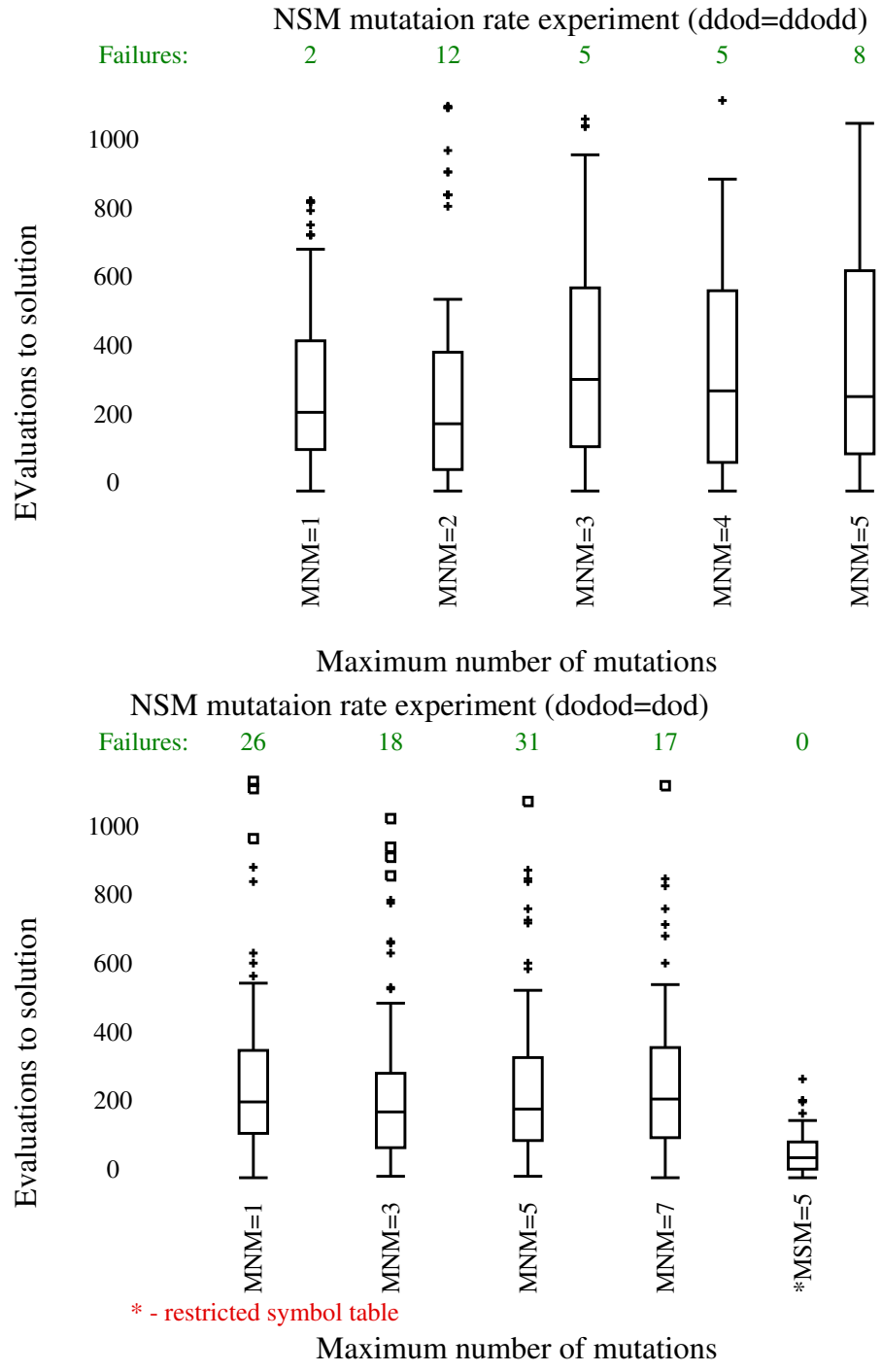


Fig. 4. Shown are the distribution of times to solution and number of runs that failed to find a problem with the requested minimum solution steps of eight. The upper panel shows the runs with initial population 1000 while the lower shows those with initial population 40.

dodod=dod	**ddod=dodd**	**ddod=ddodd**	**dodod=d**
5+5+2=6*3	31*8=6-96	22+3=66+46	1*4-9=2
3*3*6=4+7	43-6=9+62	87+8=61+44	1-9*5=2
4*8*4=9+5	55+6=9-96	97+9=24+44	9-1*5=2
3+9+3=5*5	36*8=7+73	11+7=63+63	2*4-6=1
4*4*4=7+6	65*5=3-74	27+5=88+88	6*3-9=1
7+1+7=9*9	67*9=5*51	77+6=99+39	9*4+2=1
6*6*6=5-4	54-4=3*73	53+8=72+72	3-8*1=6
9*9*9=6-5	81*5=4*93	57+5=22+29	9-1*5=2
5*5*5=6-7	88-4=8*93	58+3=27+27	4*1-9=2
1+7+7=3*3	76-7=2+82	24+4=57+55	4+2*9=1
7*5*6=3-3	64*5=8+54	22+2=87+85	2+4*9=1
6*5*6=9+9	66+5=4-84	22+2=88+79	2*7-9=1
4*4*8=5+7	82-9=6-28	88+8=44+55	2*4-6=1
9-9+5=8*2	84+2=5*58	88+7=44+61	9*4+2=1
4*4*8=7+3	42*6=8*56	33+3=14+44	9*4+2=1
3-9*8=5*4	48+3=3-66	55+5=92+42	2*4-6=1
8*8*8=3-2	48*7=9-43	83+5=19+55	2+4*9=1
3*6*3=4+7	69*8=3-16	36+3=77+58	2*7-9=1
2+4+8=9*9	31-3=6*46	66+6=17+77	2*4-6=1
3*6*6=5-8	26+8=4-84	55+5=92+97	1*9-5=2

Fig. 5. Examples of evolved, solvable puzzles.

A. Different templates yield different search behavior

The two additional templates, intended to gather more types of NSM morphing puzzles had suggestive outcomes. The template **ddod=dodd** had zero failure and the largest solution time was 45 steps. Even at a minimum search depth of 8, the problem was nearly trivial. The template **dodod=d** had 48 failures – it is a very difficult search problem.

Taken together, the results for all four templates tested suggest that the search problem is very different for different templates. It is clear that the speed differs between templates – templates with more characters generate much larger search spaces – but the algorithms were not taxed with that cost. The hard problems were allowed to run for days until the 100 runs finished.

Rather, the difficulty of the problem changed substantially from template to template. Templates with a higher proportion of digits yield easier search spaces. This is probably because the degrees of freedom to build numbers to place around the arithmetic operations is far greater when there are more digits in the template. It is also possible to write templates that strongly favor one type of operation. The template **ddod=ddd**, for example, will choose the arithmetic operation * because most true equation for that template are multiplications.

One distressing outcome that makes the softness of the parameter tuning results worse is that the change, from template to template, of the character of the search problem means that parameter tuning should be done again for each template. It may be possible to tune for templates based on their ratio of digits to arithmetic operations.

Figure 5 gives examples of twenty NSM puzzles with minimum solution length of eight edits for each of four templates. The third template is an example of a template that strongly favors a single operation. The first and fourth templates tend to find variations of the same problem; the first seems to favor repeating digits. All of these oddities support the hypothesis that different templates spawn different search problems.

VI. Conclusions and Next Steps

This study has demonstrated an effective, if somewhat inefficient, method for generating number-sentence morphing puzzles that can be solved to zero difference between the

sides of the equation in a fashion that leaves no left-over symbols and requires a specified minimum number of moves. Increasing the minimum number of moves beyond eight, a fairly generous allowance, will require improvements in the search technology.

On the other hand, it turns out that if the desired number of moves to solution is small then a majority of randomly generated examples have the property that they are puzzles that can be solved with zero difference and no left over symbols. This odd solubility phenomenon seems to arise from the combinatorial explosion of the number of configurations reachable in a small number of edits.

The is potentially some pedagogical advantage to having problems that cannot be solved, if the teacher uses the lack of a solution as a lever to create a teachable moment. The software has no problem locating problems that either cannot be brought to zero difference or can be brought to zero difference, but not using all the symbols. Figure 3 demonstrates that puzzles that are not solvable by one or another criterion are rare, and so perhaps good targets for search based PCG.

A. Fitness landscape issues

A problematic feature of this research is that the search landscapes seem to be very different from one another for different templates. It is natural for a template with more characters to create a larger search space, but the phenomena goes beyond that. The space of possible solutions to a template uses the arithmetic symbol set differently. The difficulty of the problems changes more than simple increase in size would suggest.

The issue raised in the results section of the restricted search (permitting only one arithmetic symbol in the symbol pool) being more evolution-friendly is an intriguing one. Typically when we are performing automatic content generation, the changes in the fitness landscape caused by algorithmic hacks to the fitness function do not have a visible effect on the character of the fitness landscape. It may be that the NSM search space, with its high cost fitness evaluation, makes the difference more visible.

The single operation restriction needs to be applied to additional number sentence templates to check and see if the improvement is idiosyncratic to **dodod=dod** or if this type of restriction is generally helpful. Another natural way to prune the search is to limit the total size of the symbol pool. There is a danger that such limitations might remove some search paths, but the number of moves (assuming an empty symbol pool is required in the solution) is twice the size of the symbol pool. If this modification to the search strategy becomes problematic the restriction on the size of the symbol pool could also be added to the rules for working the puzzle.

This creates a new type of research question. If we modify the representation or fitness search for a problem, to what degree does that modification induce a fitness landscape that is more friendly to evolution, or whatever search algorithm is being used. Investigating these issues is an early priority for additional research.

B. Why not speed things up with A*-search?

A*-search is a modification of dynamic programming [6] in which a progress heuristic is used to speed dynamic programming search by prioritizing which node in the search tree to expand first. The reason that A*-search is not being employed is the lack of an effective heuristic. The version of number sentence morphing appearing in [4] keeps track of the difference on both sides of the expression after every move. The game is competitive and the team(s) with the lowest difference in each turn gain bonus points. This work created, as a side effect, a clear indictment of difference between the sides of the equation as an A*-search heuristic. Along most shortest paths of most NSM problems the difference jumps up and down wildly.

The modified heuristic of permitting only one arithmetic operation in the symbol pool is similar to an A*-search restriction, but it does not yield a natural A*-heuristic. Expanding cases with the fewest arithmetic symbols first interacts badly with the need to remain well-formed and rapidly forces the deletion of arithmetic symbols as the supply of operations runs out.

C. Improving Fitness Evaluation

The elephant in the room for the current version of number sentence morphing is the speed of fitness evaluation. The vanilla version of the fitness evaluation generates the space of morphs of the number sentence under test, using dynamic programming to search for the first acceptable solution. As the algorithm satisfies the goal of needing a fairly large number of minimum steps, the size of the search trees generated explodes and the algorithm slows down.

Added to the observation that minimum solution lengths of six or more edits can be found by random sampling means that the first case were it is reasonable to use an evolutionary algorithm is one where fitness evaluation is quite expensive. The restriction on permitting at most one arithmetic symbol in the symbol table looks hopeful. The number of evaluations needed to find a solution dropped substantially, but of greater importance, the restriction of the search tree size meant that the individual fitness evaluations happened faster.

D. Representation as a target for improvement

The representation used here is a simple string in which a template enforces well formation of the initial number sentence. The by-hand technique of starting with a true number sentence and then applying edit moves to generate a NSM puzzle may be one that could be exploited to improve the evolutionary algorithm. The minimum number of steps to solve NSM puzzles generated in that fashion is bounded by the number used to generate the puzzle; it may be shorter in practice, but this technique may enrich populations with puzzles that require a large number of steps.

The authors attempted to find grammatical techniques for constructing NSM puzzles, but the fact that the actual value of the digits plays a huge part in the solution space of the puzzles caused this to be, at least so far, a fruitless venture.

E. Why no parenthesis?

It is possible to add parenthesis to number sentence morphing, at least at the level of rules, in a transparent manner. The rules remove of insert parenthesis in pairs, something that is required to maintain the well-formation of number sentences. In many cases parenthesis, inserted in a grammatically correct fashion, do not change the value of an expression and they can be inserted in a large number of places in an expression. This is the problem; adding parenthesis explodes the size of the space that is explored during fitness evaluation.

It is possible to add parenthesis to NSM puzzles like those appearing in Figure 5 without disrupting the solubility of the puzzle. It may be better to retrofit puzzles with parenthesis rather than including them in the initial search.

F. Other editing rules

The editing rules used for NSM in this study have the property that they force the player to preserve the total set of symbols used. Other editing rules are possible and may be useful for teaching other mathematical skills. One natural change is to permit the substitution of the operation symbols but not the numbers or the digits, but not the operation symbols. These puzzles would try to correct a well-formed false number sentence. The search space for such pure-substitution puzzles would be much smaller than that for number sentence morphing; one could simply write a true sentence and then perform random substitutions to generate a puzzle. The need for powerful search techniques in this study arises from the ability to insert and delete symbols together with the desire for puzzles that cannot be solved in a small number of moves.

This strategy of modifying true sentences was explored for NSM, but the path of modifications made in generating the puzzle was frequently not the shortest path to solve it.

REFERENCES

[1] D. Ashlock. Graph theory in game and puzzle design. *Game and Puzzle Design*, pages 62–70, 2016.

[2] D. Ashlock, C. Lee, and C. McGuinness. Search based procedural generation of maze like levels. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(3):260–273, 2011.

[3] D. Ashlock, C. Lee, and C. McGuinness. Simultaneous dual level creation for games. *Computational Intelligence Magazine*, 2(6):26–37, 2011.

[4] D. Ashlock and A. McEachern. Edit games. *Game & Puzzle Design*, 3(2):47–55, 2017.

[5] D. Ashlock and C. McGuinness. Landscape automata for search based procedural content generation. In *Proceedings of IEEE CIG 2013*, pages 9–16, 2013.

[6] D. Ashlock and C. McGuinness. Graph-based search for game design. *Game and Puzzle Design*, pages 68–75, 2016.

[7] Daniel Ashlock. *Evolutionary Computation for Opimization and Modeling*. Springer, New York, 2006.

[8] Jonathon Doran and Ian Parberry. Controlled procedural terrain generation using software agents. In *IEEE Transactions on Computational Intelligence and AI in Games*, volume 2, pages 111–119, June 2010.

[9] Mark Hendrikx, Sebastiaan Meijer, Joeri Van Der Velden, and Alexandru Iosup. Procedural content generation for games: A survey. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 9(1):1, 2013.

[10] Philip F. Hingston, Luigi C. Barone, and Michalewicz Zbigniew. *Design by Evolution: Advances in Evolutionary Design*. Springer Publishing Company, Incorporated, 1 edition, 2008.

[11] Lawrence Johnson, Georgios N. Yannakakis, and Julian Togelius. Cellular automata for real-time generation of infinite cave levels. In *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*, PCGames '10, pages 10:1–10:4, New York, NY, USA, 2010. ACM.

[12] A. Oliver, N. Monmarch, and G. Venturini. Interactive design of web sites with a genetic algorithm. In *IN PROCEEDINGS OF THE IADIS INTERNATIONAL CONFERENCE WWW/INTERNET*, pages 355–362, 2002.

[13] G. Smith and C. Hartveld. Procedural content generation as an opportunity to foster collaborative mindful learning. In *Proceedings of the 2014 Conference on the Foundations of Digital Gaming*, pages 1–6, 2014.

[14] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne. Search based procedural content generation: A taxonomy and survey. *IEEE Transactions on Computational Intelligence and Artificial Intelligence in Games*, 3(3):172–186, 2012.

Toward General Mathematical Game Playing Agents

Daniel Ashlock
Department of Mathematics and Statistics
University of Guelph
Guelph, Ontario, Canada
dashlock@uoguelph.ca

Eun-Youn Kim
School of Basic Sciences
Hanbat National University
Daejong, Republic of Korea
eunykim00@gmail.com

Diego Perez-Lebana
Game AI Research Group
Queen Mary University of London
London, UK
diego.perez@qmul.ac.uk

***Abstract*—General game playing AI and general video game playing AI are both active research areas. Mathematical games, like prisoner's dilemma, rock-paper-scissors, or the snowdrift game can also be played by general purpose agents. An agent representation in which agents play knowing only their own score and their opponent's score in a previous round is used to enable general mathematical game playing. Agents are trained to recognize when the game being played has changed and trained to play competently on diverse sets of games. This domain, simpler than general game playing or general video game playing, is offered as a test lab for concepts in the creation of general AI. It is demonstrated that agents can be trained to play three different coordination games competently. The agents also learn to play prisoner's dilemma and a coordination game that reverses which moves lead to high payoffs.**

***Index Terms:* general game playing, mathematical game playing, evolving agents.**

I. Introduction

General game playing AIs seek to imitate the human ability to learn and then play, at least competently, any of a broad variety of games. Any person that plays games knows that skill levels of their opponents not only vary, but vary between games. This incredibly broad mission poses a remarkably hard problem for AI training. General video game playing is a similar endeavor with a more restricted domain [20], [2]. This yielded a test domain which is more reasonable but still quite broad. Current research is driven by contests in which agents are asked to learn games they have not encountered before.

In this study it is proposed to examine *general mathematical game playing* (GMGP), an even more restricted domain. Mathematical games include games like the iterated prisoner's dilemma [14], the snowdrift game [13], and rock-paper-scissors [3]. This study will focus on simultaneous two-player games which are defined by a payoff matrix that specifies which moves are available to the players and the payoff to each player for each pair of moves.

The authors thank the University of Guelph and Canadian Natural Sciences and Engineering Research Council of Canada (NSERC) for supporting this work.

II. Background

Games are commonly used as an approximation to General Purpose AI, a field in AI in which agents must be able to respond well to multiple and unprecedented situations. They present a multitude of different scenarios, in a single or multi-player mode, in which simulations and repetitions are generally fast and available. They also capture multiple facets of character behaviour, such as hidden information, bluffing and opponent modeling, plus the complexities of random processes, as the ones determined by shuffling cards or rolling dies.

Not surprisingly, the literature shows a recent proliferation on frameworks for general game (GGP) and video game playing (GVGP). Examples are the GGP benchmark [16], the Arcade Learning Environment (ALE; [7]), OpenAI Gym [8] or the General Video Game AI (GVGAI) framework [19]. All these frameworks propose the creation of agents that should be able to play any game is given to them, often from different perspectives: single or multi player, board or video-games, via screen capture or object-based state information, etc. For this paper, we focus on the GVGAI framework, both due to its recent popularity and 2-player track.

GVGAI is a framework written in Java that proposes an interface for agents to play any of its 160 games (at the time of writing). Concretely, there are 100 single player and 60 two-player games, which have been developed over the years following its main associated competitions [20], [10]. These games are written in the Video Game Description Language (VGDL), a text based language that is able to describe two-dimensional grid arcade video games, of the like of Space Invaders, Pacman or Sokoban.

VGDL was initially proposed by Ebner *et al.* [9] at the Dagstuhl Seminar on Artificial and Computational Intelligence in Games in 2013, and then developed further by Tom Schaul [21] in a python-based framework for agent planning and learning. Games are defined using two different files: a game and a level description. While the latter specifies which sprites go where in a two-dimensional grid at the beginning of the game, the former defines four different constituents: the set of sprites (plus parameters) that take part in the game, the rules that govern their interactions, the conditions upon

978-1-5386-4359-4/18/$31.00 ©2018 IEEE

which the game ends and a mapping between characters in the level file and the sprites allowed.

Agents (also referred to as *controllers*) in GVGAI do not have access to the VGDL description of the games, but they receive (in the single and two-player planning settings) information about the current game state in two different methods that need to be implemented. The first one is a constructor, which can be used to initialize any data structure needed by the controllers, and the second one is an *act* method that is called once every frame, which requests an action to be executed by the player (or avatar) at that time step. Given that GVGAI operates in real-time, these methods must return before 1 second and 40 milliseconds respectively to avoid any penalties.

The information agents receive on each one of these methods not only contains details about the current game state (i.e. current game tick, score, avatar features and presence of other sprites), but also provides a Forward Model (FM) for controllers to roll the current state forward upon the supply of potential actions that can be returned. This FM has been used by multiple approaches in the literature [19] to develop agents based on simulation techniques such as Monte Carlo Tree Search (MCTS; [18]) or Rolling Horizon Evolutionary Algorithms [11].

For the two-player case, games use simultaneous moves, therefore the FM requires that the agent also supplies the action that (supposedly) the opponent would take. This requires, in this particular case, an effort from the agent to model the behaviour of the other player, which is still an interesting and open area of research. In their work, J. Gonzalez-Castro and Diego Perez-Leiebana [12] studied 9 different opponent models for an MCTS controller, from simple ones (assuming the opponent will use the same / the opposite action used in the last tick, or that it will use the action with the best / worst / average estimated reward) to more complex ones (using models computed over the distribution of actions used by an MCTS agent both on multiple games and during the last n frames in the current game). The authors showed that these latter approaches where able to outperform a random opponent model, which was the main model used by all top players of the competition [10].

However, further research is needed in the topic of adapting to the other player in the game. For instance, similar to the games proposed in this paper, a 20% of the 2-player games existing in the GVGAI framework are cooperative, and agents are not given the information regarding if they must compete to win or collaborate in order to achieve a common goal. Additionally, to the knowledge of the authors, no research has been undertaken in GVGAI to adapt and learn from the opponent's behaviour during the course of several repetitions of the game being played. Although the learning track of the competition [15] features agent learning via playing repetitions (with no FM), at the moment this setting is restricted to single-player games only.

The present paper attempts not only to extend the kind of games GVGAI agents can play, but also to give some initial steps in two-player opponent modeling and adaptation for general video-game playing. The agents in this study learn if they are supposed to cooperate or compete via evolutionary learning; this at least established context for alternate player modeling methods, implicitly embedded in the agent's state space by evolution.

III. Design of Experiments

A population of agents are taught to play multiple games simultaneously. This is accomplished by changing which game is being played during fitness evaluation. Agents trained on two games would play a round-robin tournament with 150 iterations of the first game, then 150 more of the second. The goal is to have the agents evolve to notice that the game has changed based only on the payoff they are receiving. The need to act based only on payoff information informs the design of the agent representation.

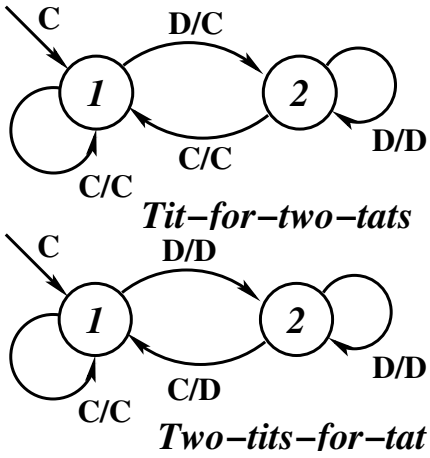


Fig. 1. Finite state agents that encode two well-know strategies for the iterated prisoner's dilemma.

Figure 1 shows two examples of finite state agents that play iterated prisoner's dilemma. These are Mealy-style machines whose transitions are driven by the opponent's last action: cooperation or defection. These agents must know that there are two moves and what those moves are. An agent trained for GMGP will not have knowledge in advance of the number or identity of moves in the game.

This design issue is addressed in this study by creating agents that based their actions on the score that they and their opponent received on the last round. Some knowledge is still required: a range in which the scores of all the games will fall.

A. The Agent Representation

The agent representation used is both relatively novel and being used in a new setting. It is a new form of, *binary decision automata* (BDA). This representation was first introduced in [17] to model stress in the workplace and

used to explore the effect of changing information resources available to game playing agents in [6]. An example of one of the BDA used in this study is given in Figure 2. These agents are Mealy architecture finite state machines. Transitions are driven by a Boolean test, in this case based on a threshold value stored in each state and the scores obtained by the agent and its opponent in previous rounds. Three tests are available: is my score larger than the threshold, is my opponent's score larger than the threshold, and is my opponent's score larger than my own. Transitions are based on true and false outcomes of these tests. The novelty of the representation is that it has no access to knowledge about what moves are being made, only what payoffs it receives. This means that the agent can detect and adapt to situations in which the rules are changed, something demonstrated in the experimental section.

State	Test	If True	If False
0:	If (His>My)	C→0;	C→1;
1:	If (My>0.27)	A→3;	A→0;
2:	If (His>My)	A→2;	A→3;
3:	If (His>3.48)	B→3;	B→0;

Fig. 2. A 4-state binary decision automata, evolved to play a game with three moves **A**, **B**, and **C**.

The agent is stored as a vector of states. The action taken for the true transition of the zeroth state is the agent's initial action, used to begin iterated play. During reproduction, two point crossover on the vector of states is used, followed by 1-MNM mutations with the number of mutations selected uniformly at random; MNM is the *maximum number of mutations*, a parameter of the representation. A mutation picks a state uniformly at random and then generate a new value for the state's threshold or one of its two transitions and actions.

In both initialization and mutation, a *shape* [5] is used that forbids any state to make a transition to itself. Preliminary work while debugging the agent software demonstrated that self-looping states that play a single move form a large local optima in the agent space. Blocking this possibility improves performance and encourages the agents to learn more complicated strategies that at least have the potential to detect and respond to a change of game.

B. The Evolutionary Algorithm

The evolutionary algorithm maintains a population of 72 agents, a number selected by preliminary experimentation to supply sufficient population diversity to learn multiple games. The agents are initialized by filling in the threshold values, actions, and transitions uniformly at random. Threshold values are selected uniformly at random in the range from the lowest to highest score an agent can receive. The agents are evolved for 250 generations, sufficient time to allow the population to stabilize genetically after burning away the agent's initial randomness.

Fitness evaluation is a round-robin iterated tournament on all pairs of distinct agents. Iterated play is continued for a number of rounds specified for each experiment. The game being played is changed during fitness evaluation so that the agent's score is the average of the scores obtained while playing multiple different games. This is one of several possible ways to expose an agent to multiple games; it was chosen because it is one of the simplest. This issue is discussed in Section V.

Selection and replacement are performed as follows. Agents are sorted into fitness order and an elite of the 48 most fit agents (two-thirds) are preserved. The twelve most fit pairs of agents are copied in fitness order. Succesive pairs of copies undergo crossover and mutation to produce 24 new agents, replacing the worst 24 in the population.

The retention of two-thirds of the agents, the *elite fraction* is a fairly important parameter. There is a substantial change in the genetic stability of an evolving population of game playing agents when this fraction is one-half or less [1]. In this study agents are given *at least* 100 rounds of play with each game with the goal of proof-of-concept that the agents can be trained to play multiple mathematical games. That informs the choice of a relatively conservative and high elite fraction of two-thirds.

C. Games Used

Experiments begin with parameter setting on sets of two and three simple coordination games. Figure 3 gives the payoff matrix for the three-move coordination games used. The first game has $X = 6$, $Y = 3$ and $Z = 1$; the second $X = 3$, $Y = 1$ and $Z = 6$; the third $X = 1$, $Y = 6$ and $Z = 3$. All of the games give a positive payoff for any coordination. The high, low, and middle payoff values are rotated to create games with substantially different payoff schemes.

Coordination Games

	A	B	C
A	X	0	0
B	0	Y	0
C	0	0	Z

Fig. 3. These matrices give the score a player making the move indexing the row of the matrix will receive against an opponent who makes the move indexing the column.

Two parameter studies are performed. The first uses the first two coordination games, the second uses all three coordination games. These games have three Nash equilibria, represented by the pure strategies. In the course of the fitness evaluation, changing which game is being played changes which of the Nash equilibria have the highest, middle, and lowest payoff.

The best of the tested parameters (eight states, $MNM = 1$) is then used for an experiment with the pair of games given

Prisoner's Dilemma or Coordination

Prisoner's Dilemma

	M_1	M_2
M_1	3	0
M_2	5	1

Coordination

	M_1	M_2
M_1	1	0
M_2	0	3

Fig. 4. These matrices give two games. The first is prisoner's dilemma, the second is a coordination game that rewards coordination on the move that corresponds to defection in the prisoner's more highly.

in Figure 4. Here fitness is evaluated on the iterated prisoner's dilemma for 150 rounds, then on a coordination game that has positive payoffs only for coordination for an additional 150 rounds. To make the problem challenging the better coordination payoff, three, is for the move that corresponds to defection in Prisoner's dilemma, while coordination on the move corresponding to cooperation in the prisoner's dilemma pays one.

D. Analysis Tools

During the course of play, the agent's average and maximum score in each generation are recorded in each generation. In the final generation, the number of each of the possible pairs of plays is recorded and the fraction of each type calculated. This creates a matrix of the following form.

$$\begin{bmatrix} 0.440 & 0.013 & 0.000 \\ 0.012 & 0.325 & 0.000 \\ 0.000 & 0.000 & 0.210 \end{bmatrix}$$

This matrix is drawn from one of the runs that achieved a high score on three games. Any off-diagonal play represents moves that gained the players no score. A good agent will place most of its moves in the diagonal entries of this matrix. In the experiments with two games, we should see similar numbers in positions (1,1) and (3,3) because these represent moves scoring the maximum possible value of 6 at some point. When three games are played we should see roughly equal distribution along the diagonal in a player that has learned to shift among the games properly.

Two statistics are extracted to check the agent's behavior. The first is the sum of the diagonal of the matrix; this is the fraction of coordinated moves or *coordination fraction*.

The second statistic uses the *Shannon entropy* of the normalized diagonal entries to estimate the evenness of distribution of the coordinated moves. This entropy is maximized by an even division among diagonal entries; the normalized diagonal will yield a score near one if divided evenly between two possibilities and 1.59 if evenly divided between three. These numbers, 1.0 and 1.59 indicate correctly learning to shift between games in agents that also have a high coordination score for the two-game and three-game coordination experiments, respectively.

E. Runs performed

A parameter study was performed for agents with 4, 6, and 8 states using mutation rates $MNM \in \{1, 3, 5\}$. These nine sets of parameters were studied for fitness evaluation with two and three coordination games. The round robin tournaments for fitness evaluation were run for 300 rounds of play. These were 150 rounds each for two games or 100 each for three games, with the games always presented in the same order and played for contiguous blocks of time.

IV. Results and Discussion

Figure 5 shows the three types of outcomes that occurred when three games were used. Unlike prisoner's dilemma, the agents in the coordination game cannot exploit one another. The only options are coordination or failure to coordinate. The costs and benefits of each of these outcomes is the same for both players. The sole issue here is the difficulty of learning to detect when the game has shifted.

The top panel in Figure 5 shows a population of agents that first discover the "just play one move" strategy. The use of the shape forbidding self-loops in the machine enables the sort of escape this population makes from that local optima to a better local optima where the payoffs are 6, 6, and 3 in the different phases of the game and then, at about generation 150, to adopt a strategy that gets a payoff of 6 in all three phases of the game. There is some inefficiency in the shift between games, and the population average score is reduced by the presence of mutant strategies, but these agents have discovered a general strategy for the three games and learned to detect a shift between the games.

The second panel of Figure 5 shows the first escape to an optima what gets payoffs of 6, 6, and 3, in some order, in the different phases of the game. This population never makes the escape to the final global optima. The third panel – one of the most common outcomes – shows agents that stay in the "play one move" local optima throughout evolution.

These exemplary runs were taken from the best performing parameters set, with the most states and the lowest mutation rate. This suggests that increasing the mutation rate makes discovery of "just play one move" strategies that use two or more states (recall one state versions are forbidden by shaping). This suggests that more complex measures might be needed to prevent this optima from being the most common outcome, but the shuffled coordination games are simple a demonstration environment in any case. This problem may not exist in more complex environments.

Figure 6 shows the fraction-of-coordination results for the two game experiments while Figure 7 shows the same quantities for the three game experiments. These results suggest more states and lower mutation rates are good. The entropy results suggest that correct shifting among the games occurred but was rare for both the two and three game experiments.

Figure 8 shows the outcomes of the experiments with a version of prisoner's dilemma and a coordination game

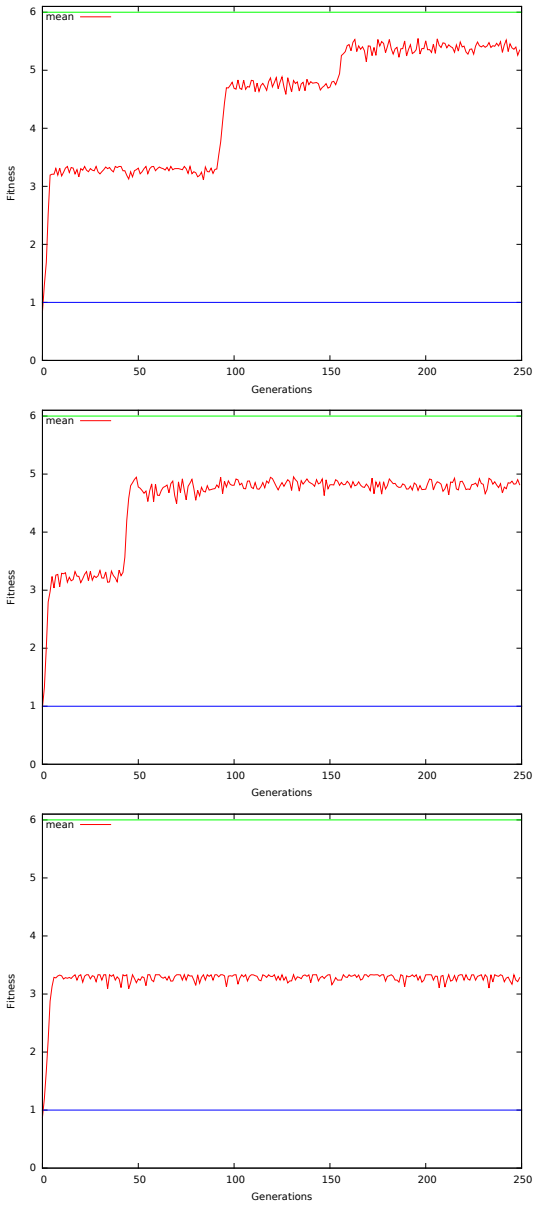


Fig. 5. Fitness over the course of evolution for three exemplary population from the runs using three games with 8 states and $MNM = 1$. The top run shows a population of agents that learned to play all three games; the middle track learned two games and accepted the second-best payoff for the third; the bottom track fails to shift games efficiently, playing a single move to get 6, 3, and 1 during different phases of fitness evaluation. The blue and green lines show the maximum and minimum values the average score can attain.

that reverses prisoner's dilemma cooperate-defect advantage structure. In thirty runs, twenty-four adopted "always move two" which corresponds to the always defect Nash equilibrium of the prisoner's dilemma and obtains the payoff of 3 from the coordination game. Three runs adopted mixed strategies. Three of the runs managed to detect the change of games and correctly shift between them.

When playing just iterated prisoner's dilemma, there is a tendency for fitness to rise and fall during the course of evolution as strategies drift away from their defensive ability though lack of use, followed by periods of exploitation. This was not visible in any of the thirty runs performed in the PD/Coordination experiments. This suggests that the need to deal with the coordination game stabilizes the agents in strategy space.

The plots in Figure 8 display the population mean and average fitness. This was done because exploitation in prisoner's dilemma corresponds to divergence of the average and maximum score. Very little such divergence occurred, even in the three runs with relatively complex mixed strategies.

V. Conclusions and Next Steps

This study presents a new representation for agents that play mathematical games and demonstrate that it can learn to play games based only on cues from scores obtained.

A. Better Training for General Agents

The fitness used to train agents for multiple games was a simple summing of the results of playing all the games involved. The games were intentionally scaled to have similar ranges of payoffs; if they had not been the lower payoff games might have been ignored. The sum-the-scores strategy makes the three problems decomposable. The agents can solve each problem individually.

In fact a few agent populations did learn to decompose the two or three games in all the collections of games used, but in all cases this was a small minority outcome. This means we gain proof-of-concept for general mathematical game playing but with a large dose of future work in improving the training algorithm.

One natural choice is multi-criteria optimization in which the score in each game is treated as a separate coordinate. This is a far more difficult training strategy, but one that seems likely to generate true general competitors across the training set.

A less complex solution to the problem consists of adding games one at a time. Permit the agent population to learn a first game, then add another game to the menu. This sort of serial addition of games may make the decomposition of the problem into distinct games more evolution-friendly.

B. Deep Time and True Generality

This study shows that the BDA-representation for agents can produce agents that recognize and correctly play games. They do not do so all that reliably and, this is more problematic, are only shown to be general across the games that serve as their training examples. A truly general mathematical game playing agent should be able to adapt to a game it has never seen before. This is clearly a domain for future work.

Hope appears in the way that agents, even when one of the games was the iterated prisoner's dilemma, tended to exhibit monotone increase in fitness. Figure 9 shows the way fitness evolves when two populations of agents are playing iterated

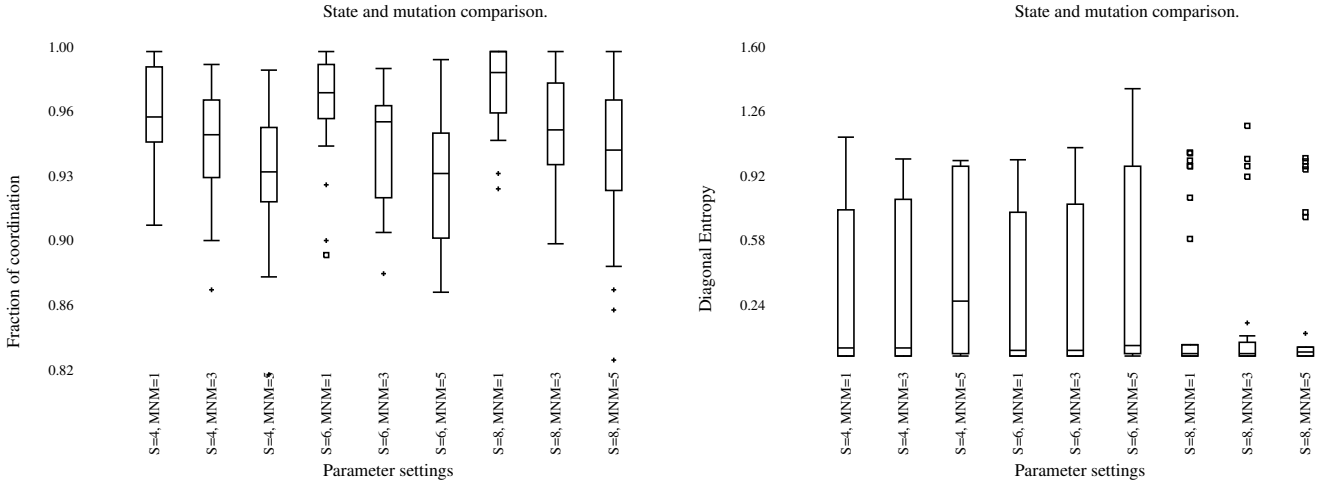


Fig. 6. Shown are the distribution of fraction of coordinated plays (left panel) and the diagonal play entropy (right panel) of populations of machines trained on play of two different coordination games.

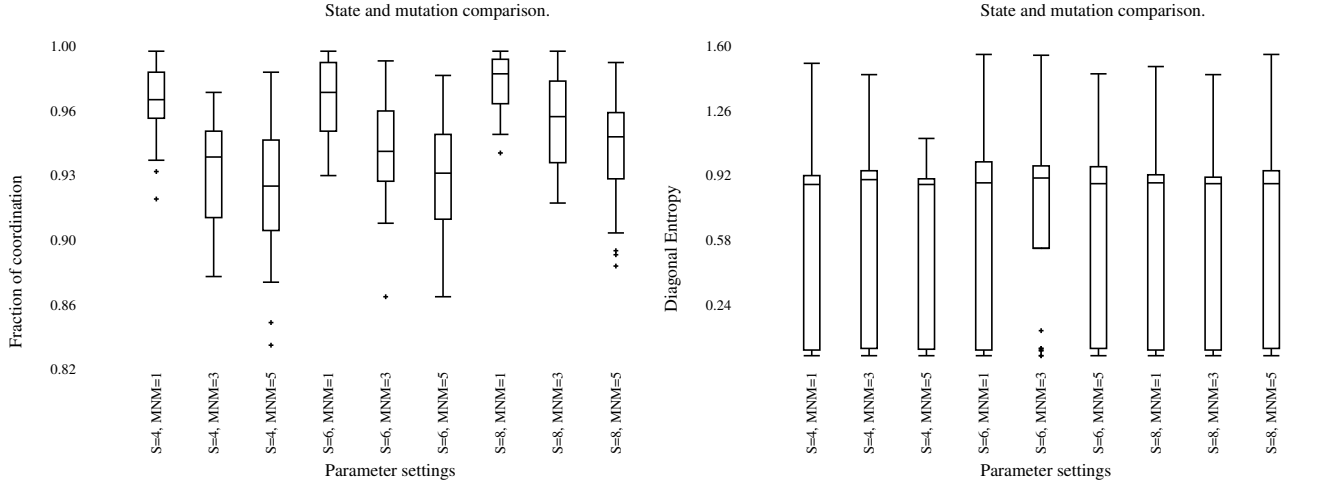


Fig. 7. Shown are the distribution of fraction of coordinated plays (upper panel) and the diagonal play entropy (lower panel) of populations of machines trained on play of three different different coordination games.

prisoner's dilemma against one another. The monotonicity of fitness suggests that serial game addition may work.

Another factor is that the number of generations, 250, used in all the simulations was chosen based on earlier work in the iterated prisoner's dilemma. Evolving for 250 generations is more than enough to generate complex behavior. In [4] new prisoner's dilemma strategies were found to arise after 32768 (2^15) generations of evolution. It may be that better performance will result from simply running evolution for a longer time.

REFERENCES

[1] Ashlock and E. Y. Kim. The impact of elite fraction and population size on evolved iterated prisoner's dilemma agents. In *Proceedings of the IEEE 2016 Congress on Evolutionary Computation*, pages 364–371, Piscataway NJ, 2016. IEEE Press.

[2] D. Ashlock, E. Y. Kim, and D. P. Lebana. General video game playing escapes the no free lunch theorem. In *Proceedings of the 2017 IEEE Conference on Computational Intelligence in Games*, pages 1–8, Piscataway NJ, 2017. IEEE Press.

[3] D. Ashlock and J. Schonfeld. Deriving card games from mathematical games. *Game and Puzzle Design*, 1(2):55–63, 2015.

[4] W. Ashlock and D. Ashlock. Changes in prisoner's dilemma strategies over evolutionary time with different population sizes. In *Proceedings of the 2006 Congress On Evolutionary Computation*, pages 1001–1008, Piscataway, NJ, 2006. IEEE press.

[5] W. Ashlock and D. Ashlock. Shaped prisoner's dilemma automata. In *Proceedings of the IEEE Conference on Computatiational Inteligence in Games*, pages 76–83, 2014.

[6] L. Barlow and D. Ashlock. Varying decision inputs in prisoner's dilemma. In *Proceedings of the 2015 IEEE Conference on Computational Intelligence in Bioioinformatics and Comutational Biology*, pages 1–8, 2015.

[7] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *J. Artif. Intell. Res.(JAIR)*, 47:253–279, 2013.

[8] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider,

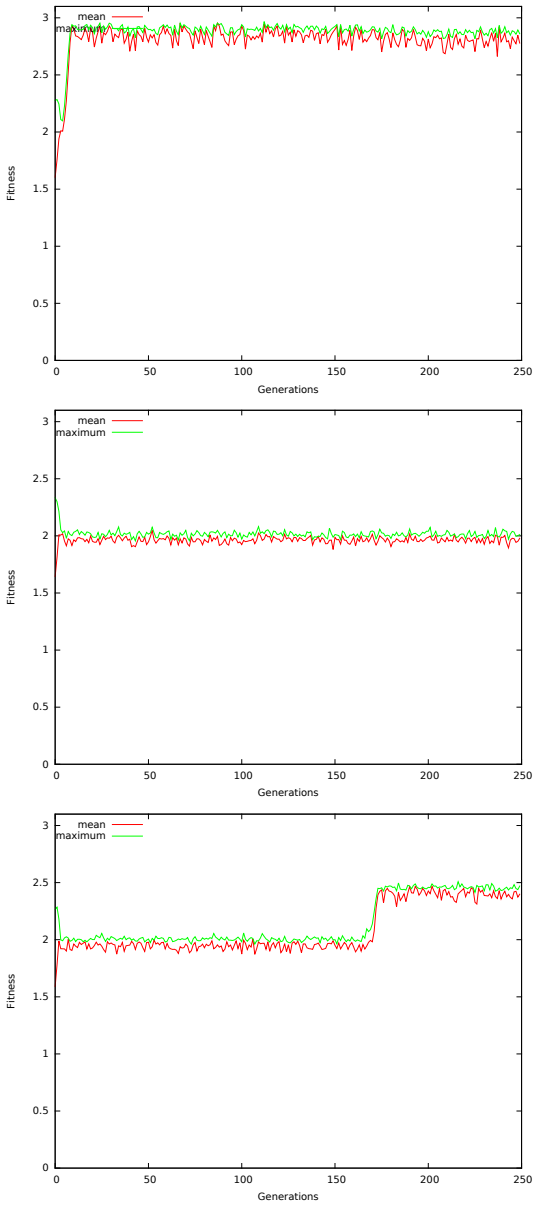


Fig. 8. Fitness over the course of evolution for three exemplary population from the runs using prisoner's dilemma and a coordination game with 8 states and $MNM = 1$. The top run shows a population of gents that learned to correctly shift between games; the middle track learned to shift games only some of the time, alternating moves in a locally optimal fashion; the bottom track fails to shift games efficiently, playing a single move to get 3 and 1 during different phases of training.

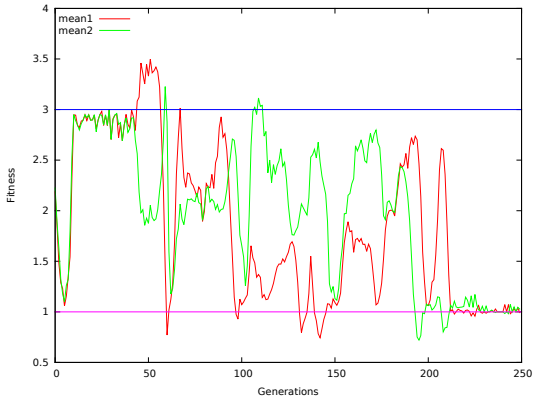


Fig. 9. An example of the non-monotone change of fitness when training agents to play prisoner's dilemma. This plot shows the average fitness of two competing populations; the blue and violet lines show the bounds that a population not engaging in active exploitation must stay between.

John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[9] Marc Ebner, John Levine, Simon M Lucas, Tom Schaul, Tommy Thompson, and Julian Togelius. Towards a video game description language. *Dagstuhl Follow-Ups*, 6, 2013.

[10] Raluca D. Gaina, Adrien Couëtoux, Dennis J.N.J. Soemers, Mark H.M. Winands, Tom Vodopivec, Florian Kirchgessner, Jialin Liu, Simon M. Lucas, and Diego Perez-Liebana. The 2016 two-player gvgai competition. *IEEE Transactions on Computational Intelligence and AI in Games*, 99:1–11, 2017.

[11] Raluca D Gaina, Simon M Lucas, and Diego Pérez-Liébana. Rolling Horizon Evolution Enhancements in General Video Game Playing. In *2017 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–8. IEEE, 2017.

[12] Jose M Gonzalez-Castro and Diego Perez-Liebana. Opponent Models Comparison for 2 Players in GVGAI Competitions. In *Computer Science and Electronic Engineering Conference (CEEC), 2017 9th*, pages 1–6. IEEE, 2017.

[13] G.W. Greenwood. Enhanced cooperation in the n-person iterated snowdrift game through tag mediation. In *Computational Intelligence and Games (CIG), 2011 IEEE Conference on*, pages 1–8, Aug 2011.

[14] J. Li, P. Hingston, and G. Kendall. Engineering design of strategies for winning iterated prisoner's dilemma competitions. *Computational Intelligence and AI in Games, IEEE Transactions on*, 3(4):348–360, Dec 2011.

[15] Jialin Liu, Diego Perez-Liebana, and Simon M. Lucas. The Single-Player GVGAI Learning Framework - Technical Manual. Technical report, Queen Mary University of London, 2017.

[16] Nathaniel Love, Timothy Hinrichs, David Haley, Eric Schkufza, and Michael Genesereth. General Game Playing: Game Description Language Specification. Technical report, Stanford Logic Group Computer Science Department Stanford University, Technical Report LG-2006-01, 2008.

[17] M. Page and D. Ashlock. Stress and productivity performance in the workforce modelled with binary decision automata. In *Proceedings of the 2015 IEEE Conference on Computational Intelligence in Bioioinformatics and Comutational Biology*, pages 1–8, 2015.

[18] Diego Perez, Spyridon Samothrakis, and Simon Lucas. Knowledge-based fast evolutionary mcts for general video game playing. In *2014 IEEE Conference on Computational Intelligence and Games*, pages 1–8. IEEE, 2014.

[19] Diego Perez-Liebana, Spyridon Samothrakis, Julian Togelius, Simon M Lucas, and Tom Schaul. General Video Game AI: Competition, Challenges and Opportunities. In *Thirtieth AAAI Conference on Artificial Intelligence*, pages 4335–4337, 2016.

[20] Diego Perez Liebana, Spyridon Samothrakis, Julian Togelius, Tom Schaul, Simon M Lucas, Adrien Couëtoux, Jerry Lee, Chong-U Lim, and Tommy Thompson. The 2014 General Video Game Playing Competition. *IEEE Transactions on Computational Intelligence and AI in Games*, 8(3):229–243, 2016.

[21] Tom Schaul. A Video Game Description Language for Model-based or Interactive Learning. In *Proceedings of the IEEE Conference on Computational Intelligence in Games*, pages 193–200, 2013.

Predicting Skill Learning in a Large, Longitudinal MOBA Dataset

M. Aung†, V. Bonometti†, A. Drachen†, P. Cowling†, A.V. Kokkinakis, C. Yoder‡, A. Wade
†Department of Computer Science, University of York
Department of Psychology, University of York
‡Riot Games
York, United Kingdom YO10 5DD and Los Angeles, ‡California 90064
Email: {mta510,vb690,anders.drachen,peter.cowling,avk504}@york.ac.uk; cyoder@riotgames.com; wade@wadelab.net

***Abstract*—The exploration of the relationships between behavior and cognitive psychology of game players has gained impetus in recent years because such links provides an opportunity for improving user experiences and optimizing products in the games industry. At the same time, the volume and global scope of digital game telemetry data has opened up new experimental opportunities for studying human behavior at large scales. Prior research has demonstrated that a relation exists between learning rates and performance. Although many factors might contribute to this correlation at least one may be the presence of innate cognitive resources, as demonstrated in recent work relating IQ and performance in a Multi-player Online Battle Arena game. Here, we extend this work by examining the relationship between early learning rate and long term performance using a 400,000 player longitudinal dataset generated by new players of the widely-played MOBA *League of Legends*. We observed that the learning rate of new players in a competitive season explains a significant amount of variance in the performance at the end of the year. This analysis was then extended by training two multivariate classifiers (Logistic Regression, Random Forest) for predicting players who by the end of the season would be considered masters (top 0.05%), based on their performance in the first 10 matches of the same season. Both classifiers performed similarly (ROC AUC 0.888 for Logistic Regression, 0.878 for Random Forest), extending the time frame for skill prediction in games based on a relatively sparse sample of early data. We discuss the implications for these findings based on preexisting psychological studies of learning and intelligence, and close with challenges and direction for future research.**

***Index Terms*—skill learning, video games, MOBA, prediction**

I. INTRODUCTION

Digital games generate considerable amounts of behavioral telemetry data [1]. However, despite the ability to track player behavior in detail within games, build detailed behavioral profiles [2] and even predict player behavior [3], behavioral analytics continues to struggle with explaining observed behaviors [4], [5]. Similarly, while work focusing on player psychology has existed for some time [6], purely psychological studies based on large-scale telemetry data are rare, in part due to difficulties in acquiring and parsing high quality, well-controlled data. In the domain of games research, there has been work exploring the correlations between player behavioral data and motivations for play [7], [8], albeit at smaller scales. Our work primarily follows recent research on skill learning and cognition in games, [9], [10], [11], [12], [13] using larger-scale data (thousands of players and upwards).

The relationship between player behavior and psychology is an ongoing research topic. Applications include designing games that are adaptive to player responses and better information modelling for AI agents [14]. Psychological research also benefits from the large datasets provided by game telemetry which increase statistical power and provide the ability to follow skill learning in individual subjects over long periods of time [9].

A growing body of evidence exists for common cognitive factors underlying early skill learning and late-stage performance, and significant progress has been made with prediction modelling based on smaller or larger scale video game data [9], [10], [11], [12]. Here the focus is on the application of this knowledge to inform classification models predicting future performance, based on data of the span of a whole season.

II. CONTRIBUTION

The work presented here contributes to the understanding of the relationship between player performance and skill learning, extending previous research on this topic. Previous work has established a correlation between skill learning and gameplay behavior[10] as well as various cognitive and motivational factors [15], [12]. However, these studies have typically utilised games designed specifically for the purpose of education or research. In this paper we explore the relationship between early learning rates and player performance after one year by analyzing datasets from more than 400,000 players of the popular commercial game *League of Legends* players during the 2016 season. All player-registered-accounts were new to the game, and were sampled randomly from a total player base of more than 100 million monthly active players as of 2016 [16]. Using this dataset, we find a strong relationship between early skill learning rates and final performance in a large-scale commercial online multiplayer game and we also present results of preliminary prediction model building. Based on previous work, we consider the possibility that this relationship is mediated by common cognitive factors [13], and propose future work to test this theory.

978-1-5386-4359-4/18/$31.00 ©2018 IEEE

III. Related work

The relationship between cognitive skills and digital gameplay has been partially explored in previous work [13], [10], [15] (see [15] for a recent review), but most research has focused on snapshot data, examining correlations between psychological factors and performance at a single point in time [17]. Longitudinal studies have tended to focus on behavior in purpose-built non-commercial games over a limited period of time. An interesting question is therefore whether cognitive resources influence skill learning and gaming performance in commercial games where the player has full control on the frequency and duration of gameplay. Recent work [15], [10], has avoided this problem by making use of behavioral telemetries from game servers. This overcomes the common issue of needing to reconstruct the acquisition process of early skill learning.

In general, digital games are an exceptional tool for studying skill learning because players can be followed and assessed from their first contact with the game. A notable example is provided by Stafford *et al.* [9], who analyzed data from 854,064 players, from an online game and established a relationship between practice volume, spacing, variability and outcome performance. However, this work was not designed as a long-term longitudinal study and made use of a relatively simple game designed specifically for the controlled purposes of that experiment. Following up on this work, a second study examined the time-series data of 20,000 players from the commercial online game *Destiny*, investigating factors that contribute to skill acquisition and learning rate [10]. Games have also been used for similar purposes by Thompson *et al.* [11], [12]. In related work, Kokkinakis *et al.* [13] provided evidence for a correlation between player performance and IQ. This work was snapshot based, i.e. based on a specific instant in time. Given the assumption that these correlations operate across any point in the learning curve of a player, the work of Kokkinakis *et al.* [13] and others, e.g. Bonny *et al.*[15] are the basis for investigating cognitive factors underlying skill learning at large scales in online games.

Related to skill learning in games is research in esports analytics focusing on predicting the outcome of player performance, either within or between matches [18], [19], [20], [21]. The majority of this work is focused on match prediction, i.e. predicting the outcome of specific matches. An example is provided by [22], who developed match win prediction models for professional-level matches in the Multi-Player Online Battle Arena (MOBA) game DOTA 2, comparing mixed-rank and professional-only rank data in terms of their applicability to a professional-level real-time prediction system. The classifier used was a hyper-parameters-tuned Random Forest model which employed a variety of in-game behavioral features as well as higher level metadata such as hero character combinations. This type of telemetry has also been used to investigate patterns of fights that occur across professional DotA 2 games [23]. Random Forest is a commonly applied model in this body of work, similar to prediction modeling work in general game analytics e.g. [24], [25].

The only current work focusing on longer-term player performance prediction in esports is an unpublished report [26] focusing on *League Of Legends*, presenting a skill prediction model for the game. However, the work is preliminary, based on 500 matches which limits the generalizability of the results. Furthermore, the work does not address the challenge of predicting the peak skill of players based on very early performance. Work such as Bonny *et al.* [15] and Kokkinakis *et al.* [13] explore longitudinal relationships between skill and behavior (or training), but do not attempt to provide prediction models.

In summary, previous work has established the foundation for behavioral prediction in games across a growing number of types and genres (e.g. [1], [21], [27], [23]), including in the MOBA genre of *League of Legends*, which is the case study used here. Work such as Stafford *et al.* [9], [10] and Thompson *et al.* [11], [12] has provided a tentative basis for exploring skill development in digital games, establishing correlations between skill learning and behavior, as well as cognition (e.g. IQ) and skill. Here we expand on this foundation.

IV. League of Legends

League of Legends is a Multi-player Online Battle Arena game (MOBA) developed by Riot Games, published in 2009. It is the most popular esports game in the world [16], [28]. The game is supported financially by microtransactions [3], [25], [24].

The game is set in an arena environment where ten players ('summoners') control 'champions', or characters in two teams of five players. Teams compete against one another to eliminate the opposing team's home base. Each match lasts approximately half an hour - although much shorter and longer matches are possible. Champions can gain more abilities during the game - primarily by accumulating 'experience points' (XP) or 'Gold' which can be used to buy performance-enhancing items. The change in Gold and XP as a function of time are two of the most commonly used performance metrics in *League of Legends*. Other important metrics in include the number of opponents that a player has killed, the number of deaths that a player has experienced (champions are 're-animated' after a time-out increasing in accordance to the champion's XP) and the number of 'assists' that one player has provided to another shortly before an enemy's death. Jointly, these metrics vary not only as a function of the player's skill and the overall skill of the two teams, but also depend on the specific champion played and the combat strategy employed. In general, *League of Legends*, similar to other MOBAs such as *DOTA 2* or *Heroes of the Storm* is conceptually simple but hard to master due to the complexity of the underlying gameplay [28].

A. *Skill and ranking in* League of Legends

Performance in *League of Legends* is calculated using an ELO-based relative skill rating system originally devised for chess. It is similar to other multi-player online games such as

Destiny [10], using a generalization of ELO called TrueSkill, a Bayesian skill rating system developed by Microsoft [29]. The system is based on wins and losses and serves the function of matchmaking, provides information to players about their rank compared to others, and can be used as a qualification for tournaments. The ELO system used by Riot is specifically adapted for the 5v5 format used in League of Legends.

Players are divided into different ranks depending on their overall skill in *League of Legends*. There are seven tiers, with the top tier being limited to 200 players. The Master rank is limited to about 0.05% of the population.

Performance is formally recorded as a hidden value Match Making Rating or Ratio (MMR). The MMR of a player is not the same thing as the rank of the player which is determined by a bin and can be influenced by additional external factors such as long periods of inactivity.

V. Data Set

A. Sample

The dataset was provided by Riot Games, the developer and publisher of *League of Legends*. The dataset contains behavioral telemetry data derived from the 2016 season of the game (ranked play in *League of Legends* is organized in game seasons of roughly one year). From the global player base, a random sample was drawn, covering 413,341 users (players) and approximately 140 million rows.

Each row in the dataset contained records for a single match in relation to a particular player account. The earliest match entry was recorded on 21 January 2016 and the final match played in the data was logged on 6 November 2016. This period of time falls under the 2016 Competitive Season of the game. All accounts had been created at the start of the season and played a minimum of 150 competitive ladder games during the season. 150 matches was set as a lower bounds to remove any largely inactive players. All matches in the data were restricted to the default 5 versus 5 ranked "Solo/Duo Queue" ranked mode. All player MMRs were initialized to the same starting value by default. After every match, this rating was then updated based on a system that takes into account the average rating of a player's team, an average rating of the enemy team, whether the player's team won or lost. Winning a match resulted in an increased rating, and a loss results in a decreased rating.

B. Telemetry

Data from League of Legends are publicly available via a data API service provided by Riot. The dataset provided here was similar in to the data that might be acquired from these public sources but had the advantage of being randomly drawn from the population, and, critically, contained Rating (MMR) scores for each player, which are not publicly available. 16 features were provided for this analysis (Table I).

TABLE I
Raw In-Game Data, on a per match basis

Field	Description
Account ID	Unique Identifier of a player account
Platform ID	The server the game was played on
Game ID	Unique identifier of the match
Neutral Creep	Number of neutral AI enemies killed
Enemy Creep	Number of AI enemies killed
Win	Boolean indicating a win or loss
Timestamp	When the match was logged
Date	Date of match played
Hour	Hour of match played
Gold Earned	Total gold earned in the match
Damage Dealt	Total dealt to other players
Time Dead	Total seconds spent dead
Time Played	Total seconds played in the match
Kills	Total Kills
Deaths	Total Deaths
Assists	Total Assists
Rating	The rating of the player before the match
Position	The role the player was assigned

The raw data forms the basis for feature engineering (see also Table II).

VI. Methods

Data were preprocessed and analyzed in a Python 3.6 environment using Pandas [30], Numpy and SciPy [31] for data handling and statistical analysis. Scikit-learn [32] was the reference framework for machine learning.

A. Data Preprocessing

The data provided by Riot were drawn directly from the telemetry servers of League of Legends. In addition to imposing a 150 game minimum requirement, and season time bin we performed some additional pre-processing steps to ensure data quality.

We first eliminated players playing more than 3,000 matches in total, as we were cautious of excessive playtimes indicating possible contamination from shared accounts or automated systems. Following this, we also filtered players whose first MMR entry differed from the pre-defined starting value. This was determined to be an artifact caused by players migrating between different servers during the season, displaying only records of their play on their latest server in our data. We then eliminated player who abandoned (went 'away from keyboard' early in the game) during the first 10 games they played because scores during this period were a critical component of our analysis. To eliminate these players we filtered users that recorded Time Played durations of less than 900 seconds. Despite a legitimate match in LoL might be shorter, these extremely rare cases hold very little information about the player performance and are indistinguishable from those where the player decided to leave the game. We also discarded users who recorded simultaneous Kills, Deaths and Creep Kill scores of 0 for the same reason. Finally we also excluded users having their nominally unique id duplicated on multiple servers.

From the original data of 413,341 players, 313,184 were retained after preprocessing. Standardized distributions of MMR for this sample can be observed in figure 1, and the trajectories of MMRs over the season can be observed in figure 2.

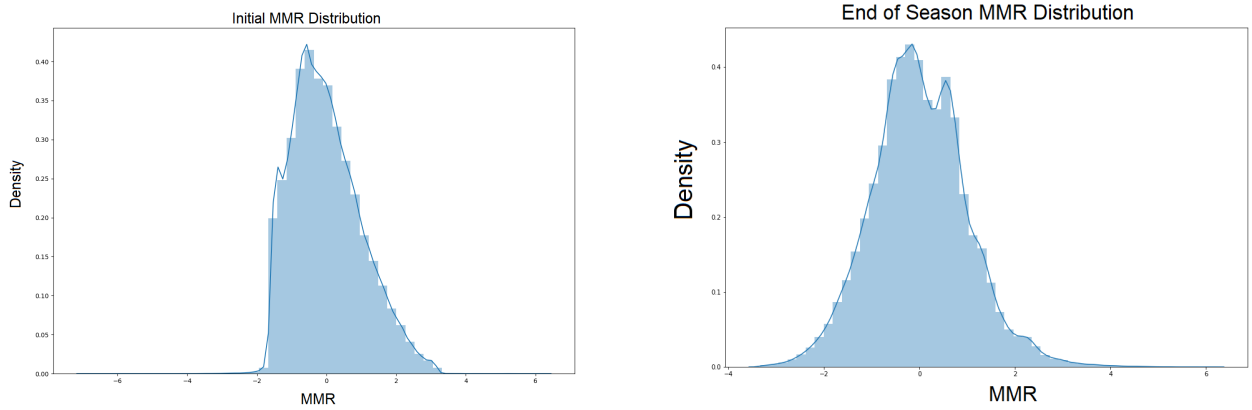


Fig. 1. Distributions of players by MMR during calibration games (left) and after the season end (right). Due to the confidential nature of the MMR values, the axes have been standardized.

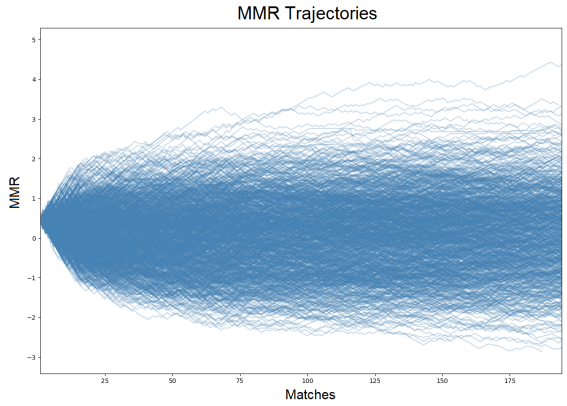


Fig. 2. Trajectories of MMRs over matches from a subsample of players. All the trajectories stem from the same starting point and spread in the initial stages mirroring a power law curve. Due to the confidential nature of the MMR values, the axes have been standardized.

B. Regression Analysis

For each region we evaluated whether the rate of change in the MMR of the first 10 matches predicted the mean MMR of the last 10 matches. For comparison, we generated a synthetic null data set by computing 100,000 random walks with length and MMR transition probabilities drawn at random from distributions matching the existing data.

C. Feature Engineering

Since the aim of this work was to evaluate the impact of early season performance on final season outcome we computed a set of features based on the original Key Performance Indices (KPIs) over the first 10 matches of each user (again see Table 1 for further details). Two approaches were adopted: a brute force one where we retrieved various statistical descriptors of the original KPIs and an informed one where we used knowledge derived from our regression analysis and previous work [10], [27], [24], [25], [12] for retrieving possible useful features.

In first instance a series of temporal KPIs were created based on the in-game time alive (i.e. Time Played - Time Dead): Neutral Creep per Minute, Enemy Creep per Minute, Gold per Minute, Damage per Minute, Kills per Minute, Deaths per Minute and Assists per Minute. For each of the original and temporal KPIs we retrieved mean, median and standard deviation over the first 10 matches in accordance to the methodology found in [33]. Following the intuition of [10] we computed a series of progression metrics retrieving the first derivative obtained by regressing a particular KPI over the ordered number of matches (i.e. range from 1 to 10). We calculated the first derivative for: Gold, Damage Dealt, Time Alive, Time Dead, Kills, Deaths, Assists, Gold per Minute, Deaths per Minute, Assists per Minute and MMR over the first 10 games.

To avoid problems of instability when calculating ratios with a denominator close to zero we computed the percentage for the following sets: {Time Alive, Time Dead}, {Deaths, Kills, Assists}, {Win, Loss}, {Morning Session, Afternoon Sessions, Evening Sessions, Night Sessions}, and {Position Utility, Position Middle, Position Bottom, Position Top, Position Jungle}.

We also calculated a series of miscellaneous features like Mean Temporal Distance between matches, number of Consecutive Wins, number of Consecutive Losses and variability in the role assumed by the player as measured by the Gini Index. These were added because previous work has utilized these metrics e.g. for prediction modeling in games or to explore skill learning (e.g. [10], [27], [24], [25], [12]

As target variables for our regression analysis and multivariate classification task we retrieved for each user the mean MMR over the last 10 matches and the difference between the mean MMR of first and last 10 matches. A summary of features generated can be observed in Table II.

TABLE II
SUMMARY OF THE FEATURES ENGINEERED FROM ORIGINAL RAW DATA.

Feature Type	Description	Example
Statistics	Common statistical descriptors	Mean kills Median kills Standard deviation Kills
Progression	Rate of change over matches	First derivative of kills over first 10 matches
Percentages	Percentages over particular sets of raw data	Percentage of Kills Percentage of Deaths Percentage of Assists
Miscellaneous	Features covering specific aspects of the game	Role variability Mean temporal distance between matches
Targets	Metric employed for regression and classification	Mean MMR over the last 10 matches (Mean MMR last 10 matches) - (Mean MMR first 10 matches)

D. Predictive Skill Modeling

Despite an interesting goal would have been forecasting the players' performance in continuous fashion (i.e. regression), we decided to focus on an early detection of extremely proficient players (i.e. classification)[34]. This solution allowed us both to maximize the prediction power and to provide a starting point for addressing issues relevant for the competitive games industry (i.e. player scouting). For our purpose we used two common machine learning algorithms, Logistic Regression (LogReg) and Random Forest (RanFor) [35], able to capture both linear and non linear interactions between the features. We chose these algorithms because despite their simplicity, they can often achieve good result while still providing useful insights (i.e. visualization of features importance). Furthermore, these are models heavily used in game analytics research for prediction tasks (see e.g. [27], [18], [24]).

In first instance we created a labeling system for differentiating the best performing users from the rest of our sample. The metric employed for this labeling system was the difference between the mean MMR of the first 10 matches and the mean MMR of the last 10 matches, this has been done for avoiding that informations contained in the input features (derived from the first 10 matches' KPIs) leaked in the metric employed for creating the labels consequently biasing the classification model. Nevertheless, for transparency reasons, we also conducted the same classification task employing labels derived from the mean MMR of the last 10 matches but due to space constrains the relative results are reported exclusively in table IV. The labeling system employed a percentile based encoding where all the players below the 99.95 percentile were encoded as negative samples while all the others as positive. We then divided the original data-frame in validation (n = 209,834) and test set (n = 103,350) via Stratified Shuffle Split [32]. This was essential given the extreme imbalance in the label distribution. We used the validation set for searching for optimal hyper-parameters and the test set for performing the final prediction. For each model the best combination of hyper-parameters was found by using a Grid Search 10 Fold Stratified Shuffle Cross Validation and selecting the best model based on the average ROC AUC score. Since the labels distribution was extremely imbalanced for avoiding under or over-sampling our dataset we applied a weight to each label inversely proportional to its frequency in the input data [32].

To improve the performance of the Logistic Regression and allowing the interpretation of the coefficients associated to each feature, when using this model we rescaled the features using a method that is robust to outliers (i.e. removing the median and rescaling the data accordingly to the quantile range). After tuning the hyper-parameters to discover the best model we retrieved the top 20 features contributing the most to the classification performance, although this can provide insights, given the high inter-correlation between our features caution has to be posed in the interpretation of their importance.

VII. RESULTS

A. Regression Analysis

The learning rate computed from the first 10 games was correlated significantly with the final average performance level (fig. 3). This correlation achieved significance across all servers with p values less than .0001 in all case. Effect sizes (r2) ranged from .25 to .37. Performance improved with the number of initial games chosen with an approximately linear dependence up to 40 games. As expected, our randomized control dataset using a large set of simulated players (n=100,000) also exhibited a statistically significant relationship between slope and final score ($p<.0001$) reflecting the fact that a slight positive slope in the initial stages of a random walk will tend, on average, to result in a slightly positive final value. However, in this case the effect size was very small (r2 = .008). Similar results were observed in each server independently (Table III).

TABLE III
REGRESSION RESULTS

Set	r2	p value
North Europe	0.308	<.0001
West Europe	0.378	<.0001
Brazil	0.297	<.0001
Latin America 1	0.257	<.0001
Latin America 2	0.307	<.0001
Oceania	0.353	<.0001
North America	0.374	<.0001
Japan	0.315	<.0001
Total sample	**0.345**	**<.0001**
Random Walk	**0.008**	**<.0001**

B. Predictive Skill Modelling

The best hyper-parameters found by the grid search for the Logistic Regression were L1 penalty with inverse regularization equal to 0.01 while those for the Random Forest included entropy as a split evaluation metric, maximum depth of the tree equal to 10, maximum number of features employed by each

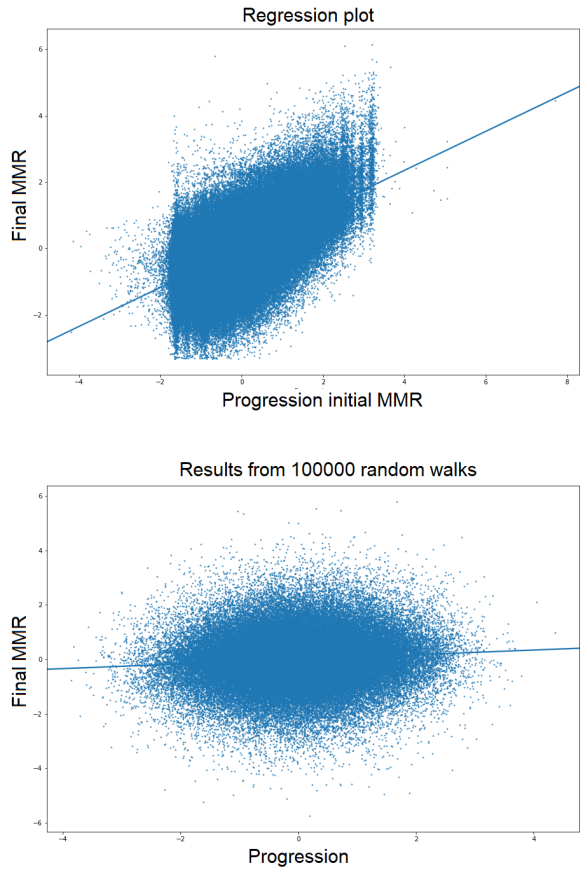


Fig. 3. Regression plots for total sample learning rate and final MMR (top), and random walk synthetic set (bottom). Due to the confidential nature of the MMR values, the axes have been standardized.

tree equal to the square root of the total number of features, maximum number of leaf nodes equal to 15 and number of trees populating the forest equal to 60. As mentioned before, we only took into account the results derived from the adoption of the difference based labeling system, however, for visibility purposes, in table IV we also reported the results from the alternative labeling system. The fields in Table IV specify the model employed, the metric on which the labeling system is based, the weighted f1 score (i.e. accounting for imbalance in the labels distribution), the ROC AUC score, the number of true positive, true negative, false positives and false negatives. For a better overview of the models' performances we computed and plotted normalized confusion matrices showing the percentages of correct and incorrect classifications (fig. 4) as well as bar charts showing the top 20 features contributing the most in the classification task (fig. 5).

VIII. Discussion

We find that the initial rate of MMR change is strongly related to the final end-of-season MMR in League of Legends. This suggests that a common factor which we identify as cognitive performance underlies learning and performance in

TABLE IV
Results of Predictive Analyses

Model	Metric	f1	AUC	TN	FN	TP	FP
LogReg	**Diff**	**0.940**	**0.888**	**92,077**	**61**	**456**	**10,757**
RanFor	**Diff**	**0.938**	**0.878**	**91,780**	**70**	**447**	**11,054**
LogReg	Final	0.953	0.923	94,534	37	480	8,300
RanFor	Final	0.952	0.923	94,309	36	481	8,525

Results of Logistic Regression and Random Forest for prediction using either end of season MMR (Final) or MMR change (Diff). AUC: area under ROC, TN: true negative, FN: false negative, TP: true positive, FP: false positive

this game - and likely in other similar MOBAs. Our results build on the finding of by Dewar and Stafford [9], extending them in several ways. Dewar and Stafford's data were obtained from users playing a non-commercial on-line game specifically designed for educational and research purposes. In this respect their findings mirror those of Quiroga *et al.* [36] who used custom-made game-like tests to probe IQ. In comparison, the game we analyze here is a commercial product with an active user-base that numbers in the hundreds of millions. The statistical findings we present are therefore extremely robust due to the sample size used, and of general interest because of their ecological relevance. We also presented promising results from a multivariate classification task showing that end of season exceptional performance can be identified employing metrics derived from the early matches.

Our results support a growing body of work indicating that cognitive performance [13], [10], [15] and possibly other psychological factors [7], [8] are exposed by game telemetry. This observation can be used in at least two ways. First, it has significant value to psychological research as it provides a way of evaluating cognition at the population level in real-time and at large scales. In previous work [13], we have raised the possibility of large-scale video game data being used to perform 'cognitive epidemiology' - a population-level assessment of cognitive health which might provide an early indicator of environmental changes (for example, disease, pollution or social factors) that affect cognition. Second, it is of interest to the e-sports analytics community because it provides a theoretical basis for performing longitudinal game analytics - allowing analysts to predict, for instance, churn rate, future performance levels, and potentially complex player-player interactions.

Our future work is focused on further exploring the link between video game data and psychological factors. While cognitive performance is important, it is just one of a wide range of psychological factors that can be extracted from these rich datasets. We expect that these factors will, like the one studied here, provide insights into psychology at a global level while also providing the games industry with theoretically validated tools to improve their products and user experience.

IX. Limitations and Future Works

We acknowledge that focusing our work on a single MOBA title might pose limitations to the generalisability of the results.

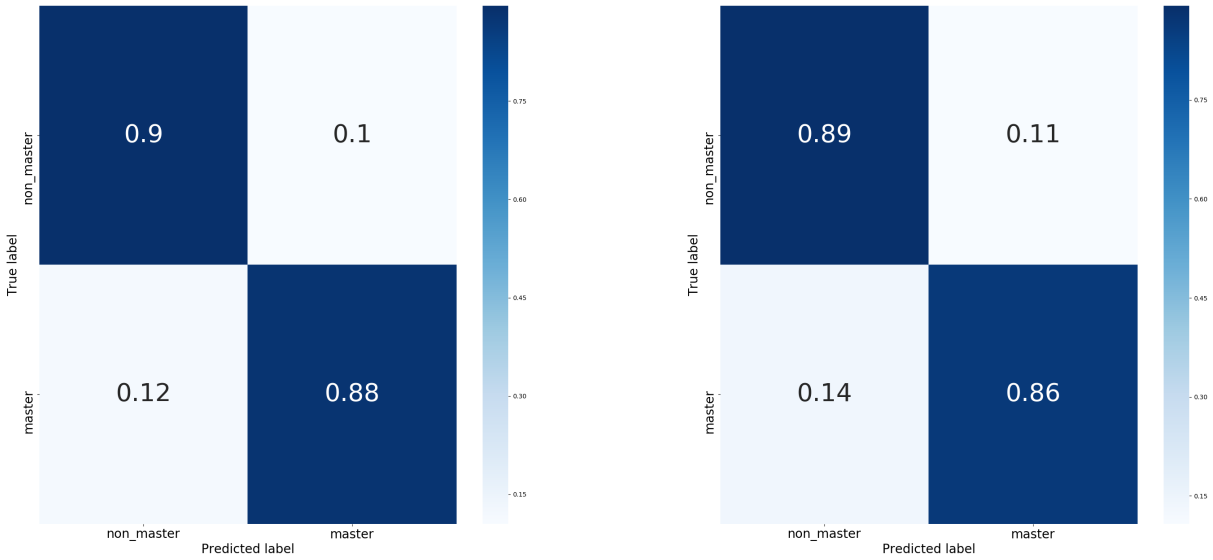


Fig. 4. Normalized Confusion Matrices for logistic regression difference based labeling system (top) and random forest difference based labeling system (bottom).

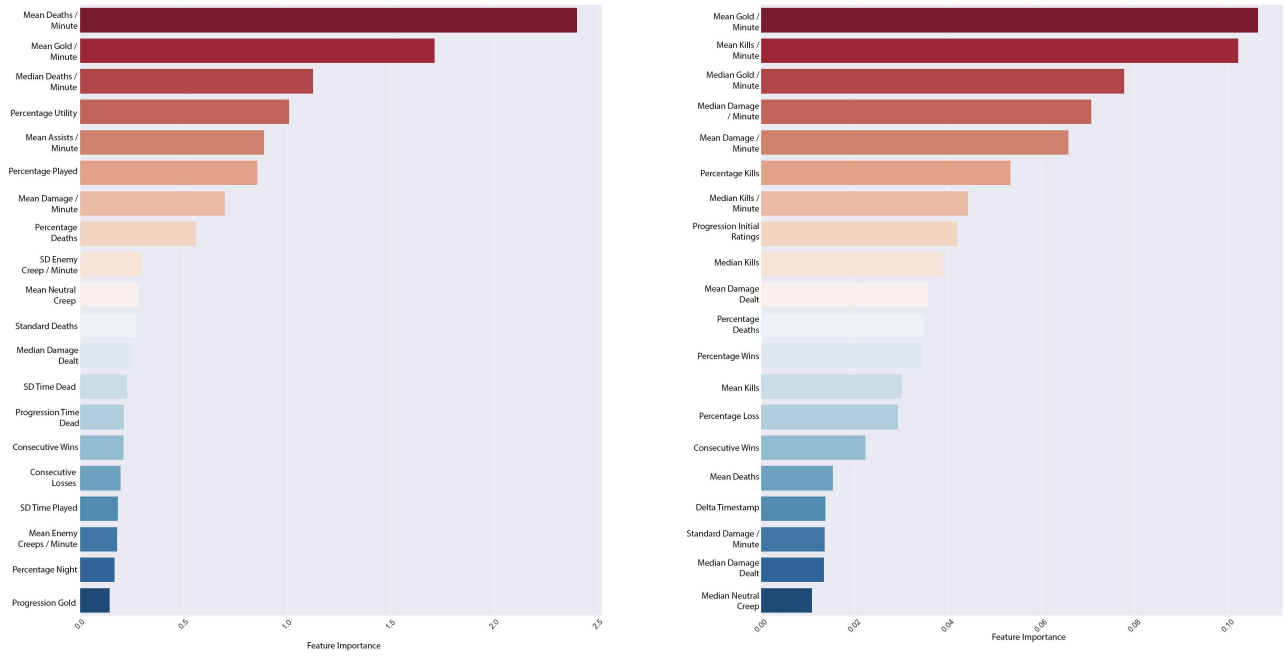


Fig. 5. Feature importance: 20 most important features for logistic regression (left) and random forest classifier (right). The two models identify different sets of features as the most important predictors. However, the top five features for both models all deal with player deaths, gold gain and player kills and damage dealt. Logistic regression adds in the percentage of time spent playing utility roles also. Looking at the top 20 predictors, there is some difference between the two models but both include similar feature sets: kills, deaths, damage and gold. Notably, win and loss conditions feature relatively low on the features ranking (”consecutive wins” placed 13th for both models). This indicates that the win/loss features are perhaps too aggregate (i.e. encapsulating performance of both teams in the game) to be highly significant predictors of individual skill/performance.

These types of analysis are complicated by the requirement to gain access to raw ELO scores for these game. This often requires licencing agreements with the companies that are not straightforward to obtain. Nevertheless, one possible direction for future work would be to attempt to replicate the results presented here employing data from different games.

X. Declaration of Conflicting Interests

The authors declare that they have no conflicting interests.

XI. Acknowledgements

We thank Riot Games for providing the data used in this work. We would also like to thank Sagarika Patra for her help and advice with statistics and data modeling. Part of this work was conducted in the Digital Creativity Labs (www.digitalcreativity.ac.uk), jointly funded by EPSRC/AHRC/InnovateUK under grant no EP/M023265/1.This work was also supported by grant EP/L015846/1 for the Centre for Doctoral Training in Intelligent Games and Game Intelligence (IGGI - http://www.iggi.org.uk/) from the UK Engineering and Physical Sciences Research Council (EPSRC).

References

[1] M. Seif El-Nasr, A. Drachen, and A. Canossa, *Game Analytics - Maximizing the Value of Player Data*. Springer, 2013.

[2] A. Drachen, R. Sifa, C. Bauckhage, and C. Thurau, "Guns, swords and data: Clustering of player behavior in computer games in the wild," in *IEEE Conference on Computational Intelligence and Games*. IEEE, 2012, pp. 163–170.

[3] M. Milosevic, N. Zivic, and I. Andjelkovic, "Early churn prediction with personalized targeting in mobile social games," *Expert Systems with Applications*, 2017.

[4] A. Drachen and S. Connor, "Game analytics for games user research," in *Games User Research*. Oxford University Press, 2018, pp. 333–354.

[5] A. Drachen, J. Green, C. Gray, E. Harik, P. Lu, R. Sifa, and D. Klabjan, "Guns and guardians: Comparative cluster analysis and behavioral profiling in Destiny," in *IEEE Conference on Computational Intelligence and Games*. IEEE, 2016, pp. 1–8.

[6] N. Yee, "The demographics, motivations, and derived experiences of users of massively multi-user online graphical environments," *Presence: Teleoper. Virtual Environ.*, vol. 15, no. 3, pp. 309–329, 2006.

[7] A. Canossa, J. B. Martinez, and J. Togelius, "Give me a reason to dig minecraft and psychology of motivation," in *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*. IEEE, 2013, pp. 1–8.

[8] G. Van Lankveld, P. Spronck, J. Van den Herik, and A. Arntz, "Games as personality profiling tools," in *Computational Intelligence and Games (CIG), 2011 IEEE Conference on*. IEEE, 2011, pp. 197–202.

[9] T. Stafford and M. Dewar, "Tracing the Trajectory of Skill Learning With a Very Large Sample of Online Game Players," *Psychological Science*, vol. 25, no. 2, pp. 511–518, Feb. 2014. [Online]. Available: http://journals.sagepub.com/doi/10.1177/0956797613511466

[10] T. Stafford, S. Devlin, R. Sifa, and A. Drachen, "Exploration and Skill Acquisition in a Major Online Game," in *The 39th Annual Meeting of the Cognitive Science Society (CogSci)*. York, 2017.

[11] J. J. Thompson, M. R. Blair, L. Chen, and A. J. Henry, "Video game telemetry as a critical tool in the study of complex skill learning," *PloS one*, vol. 8, no. 9, 2013.

[12] J. J. Thompson, C. M. McColeman, E. R. Stepanova, and M. R. Blair, "Using video game telemetry data to research motor chunking, action latencies, and complex cognitive-motor skill learning," *Topics in Cognitive Science.*, vol. 9, no. 2, pp. 467–484, 2017.

[13] A. V. Kokkinakis, P. I. Cowling, A. Drachen, and A. R. Wade, "Exploring the relationship between video game expertise and fluid intelligence," *PLOS ONE*, vol. 12, no. 11, p. e0186621, Nov. 2017. [Online]. Available: http://dx.plos.org/10.1371/journal.pone.0186621

[14] G. N. Yannakakis, "Game ai revisited," in *Proceedings of the 9th Conference on Computing Frontiers*. IEEE, 2012, pp. 285–292.

[15] J. W. Bonny and L. M. Castaneda, "Number processing ability is connected to longitudinal changes in multiplayer online battle arena skill," *Computers in Human Behavior*, vol. 66, pp. 377–387, 2017.

[16] "Riot games reveals league of legends has 100 million monthly players," *Forbes*, 2016. [Online]. Available: https://www.forbes.com

[17] A. D. Castel, J. Pratt, and E. Drummond, "The effects of action video game experience on the time course of inhibition of return and the efficiency of visual search," *Acta Psychologica*, vol. 119, pp. 217–230, 200.

[18] J. Kawale, A. Pal, and J. Srivastava, "Churn prediction in MMORPGs: A social influence based approach," in *International Conference on Computational Science and Engineering*, vol. 4. IEEE, 2009, pp. 423–428.

[19] H. Xie, S. Devlin, D. Kudenko, and P. Cowling, "Predicting Player Disengagement and First Purchase with Event-frequency Based Data Representation," in *Proc. of CIG*, 2015.

[20] M. Schubert, A. Drachen, and T. Mahlmann, "Esports analytics through encounter detection," in *Proceedings of the MIT Sloan Sports Analytics Conference*, 2016.

[21] F. Rioult, J.-P. Metivier, B. Helleu, N. Scelles, and C. Durand, "Mining tracks of competitive video games," *AASRI Procedia*, vol. 8, pp. 82–87, 2014.

[22] V. Hodge, S. Devlin, N. Sephton, F. Block, A. Drachen, and P. Cowling, "Win Prediction in Esports: Mixed-Rank Match Prediction in Multi-player Online Battle Arena Games," *arXiv preprint arXiv:1711.06498*, 2017.

[23] P. Yang, B. E. Harrison, and D. L. Roberts, "Identifying patterns in combat that are predictive of success in MOBA games." in *FDG*, 2014.

[24] J. Runge, P. Gao, F. Garcin, and B. Faltings, "Churn Prediction for High-value Players in Casual Social Games," in *Proc. of IEEE CIG*, 2014.

[25] F. Hadiji, R. Sifa, A. Drachen, C. Thurau, K. Kersting, and C. Bauckhage, "Predicting player churn in the wild," in *Computational intelligence and games (CIG), 2014 IEEE conference on*. IEEE, 2014, pp. 1–8.

[26] J. Min, S. W. Jang, and J. Ha, "Predicting Matchmaking Rating (MMR) Change in League of Legends."

[27] R. Sifa, F. Hadiji, J. Runge, A. Drachen, K. Kersting, and C. Bauckhage, "Predicting purchase decisions in mobile free-to-play games," in *Proc. Artificial Intelligence and Interactive Digital Entertainment International Conference*, 2015, pp. 79–85.

[28] Y. J. Kim, D. Engel, A. W. Woolley, J. Y.-T. Lin, N. McArthur, and T. W. Malone, "What Makes a Strong Team?: Using Collective Intelligence to Predict Team Performance in League of Legends." ACM Press, 2017, pp. 2316–2329. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2998181.2998185

[29] "Trueskill: A bayesian skill rating system."

[30] W. McKinney, "pandas: a foundational python library for data analysis and statistics," *Python for High Performance and Scientific Computing*, pp. 1–9, 2011.

[31] E. Jones, T. Oliphant, P. Peterson *et al.*, "SciPy: Open source scientific tools for Python," 2001–, [Online; accessed ¡today¿]. [Online]. Available: http://www.scipy.org/

[32] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and . Duchesnay, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, no. Oct, pp. 2825–2830, 2011. [Online]. Available: http://www.jmlr.org/papers/v12/pedregosa11a.html

[33] S. Kim, D. Choi, E. Lee, and W. Rhee, "Churn prediction of mobile and online casual games using play log data," *PloS one*, vol. 12, no. 7, p. e0180735, 2017.

[34] "Rank distribution: League of legends," *League of Graphs*, 2008. [Online]. Available: https://www.leagueofgraphs.com/rankings/rank-distribution

[35] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[36] M. A. Quiroga, S. Escorial, F. J. Romn, D. Morillo, A. Jarabo, J. Privado, M. Hernndez, B. Gallego, and R. Colom, "Can we reliably measure the general factor of intelligence (g) through commercial video games? Yes, we can!" *Intelligence*, vol. 53, pp. 1–7, Nov. 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S010280961500104X

An Eye Gaze Model for Controlling the Display of Social Status in Believable Virtual Humans

Michael Nixon
Institute of Communication, Culture, Information and Technology
University of Toronto
Toronto, Canada
michael.nixon@utoronto.ca

Steve DiPaola
School of Interactive Arts and Technology
Simon Fraser University
Vancouver, Canada
sdipaola@sfu.ca

Ulysses Bernardet
School of Engineering and Applied Science
Aston University
Birmingham, United Kingdom
u.bernardet@aston.ac.uk

***Abstract*—Designing highly believable characters remains a major concern within digital games. Matching a chosen personality and other dramatic qualities to displayed behavior is an important part of improving overall believability. Gaze is a critical component of social exchanges and serves to make characters engaging or aloof, as well as to establish character's role in a conversation.**

In this paper, we investigate the communication of status related social signals by means of a virtual human's eye gaze. We constructed a cross-domain verbal-conceptual computational model of gaze for virtual humans to facilitate the display of social status. We describe the validation of the model's parameters, including the length of eye contact and gazes, movement velocity, equilibrium response, and head and body posture. In a first set of studies, conducted on Amazon Mechanical Turk using pre-recorded video clips of animated characters, we found statistically significant differences in how the characters' status was rated based on the variation in social status.

In a second step based on these empirical findings, we designed an interactive system that incorporates dynamic eye tracking and spoken dialog, along with real-time control of a virtual character. We evaluated the model using a presential, interactive scenario of a simulated hiring interview. Corroborating our previous finding, the interactive study yielded significant differences in perception of status were found ($p = .046$). Thus, we believe status is an important aspect of dramatic believability, and accordingly, this paper presents our social eye gaze model for realistic procedurally animated characters and shows its efficacy.

***Index Terms*—procedural animation, believable characters, virtual human, gaze, social interaction, nonverbal behaviour, video games**

I. Introduction

Designing good believable characters to inhabit virtual story worlds is an ongoing problem, and remains a major concern within digital games. Researchers have been investigating many different methods for creating believable expressive character that can dynamically adapt either at design-time to their authors' needs, or during run-time as the situation they find themselves in evolves. The field of believable characters deals with the intersection of visual aesthetics and intelligent agent research. In general terms, intelligent agents are computer systems that possess autonomy, social ability, reactivity, and pro-activeness [1], and are conceptualized and implemented using anthropomorphic terms, including beliefs, desires, intentions [2], and other types of cognitive models. Adding these models allows agents to participate in ongoing stories, as they possess enough cognitive depth so as to be interesting. Additionally, as the agents are visually represented as a human character, they have to meet a range of expectations for them to appear lifelike and interesting.

Believable character research focuses on providing embodied intelligent agents with overt human bodies and corresponding characteristics for various purposes including entertainment (e.g. Façade [3],), training (e.g. the Virtual Humans [4] project), and education (e.g. Fear Not! [5]).

To be believable, these characters' bodies need the ability to communicate using both verbal and non-verbal channels simultaneously. The synchronization of these channels is vital to meeting appropriate expectations from human communication, reducing the potential Uncanny Valley effect and making the computer less of an "unfamiliar interlocutor" [6]. This means such improvements help in a wide range of use cases where appearing human is desirable, including games. Along with accompanying communication, it is important that characters can use non-verbal behavior to send social signals, which are "...complex aggregates of behavioral cues accounting for our attitudes" [7]. This allows them to participate in more dramatically interesting scenarios. Status is one of these important signals, defined as an embodied or felt sense of one's social position relative to others.

We chose status as the aspect to investigate since it is considered very impactful in the performing arts, and thus will be valuable in expressive virtual performances, including those of NPCs in digital games. Status was first popularized in improv and interactive theater by Johnstone [8], and is now considered a core concept [9], useful for driving dramatic activity. This further builds on status as a signal sent by behaviour rather than overt symbols of e.g. wealth.

Gaze plays a central role in nonverbal communication; empirical research has found high rates of correlation between eye gaze and specific conversational actions [10]. Furthermore, these expectations persist even with androids [11] and seem to underlie the human ability to understand one another; violating

This research was supported by the Social Sciences and Humanities Research Council of Canada.

them can lead to the Uncanny Valley [12] effect, where viewers find characters eerie or inhuman. Besides raw technical improvements to rendering and resolution, modeling agent behavior after expected human behavior provides another way of increasing their believability. As well as enhancement, gaze can function along with other body language to provide contrasting messages or subtle meanings to verbal phrases. We believe this can be synthesized with observations from performance art to allow us to incorporate powerful dramatic concepts that draw on embodied experience. Overall, concern for the gaze of characters, which impacts social signaling and conversational cues, implies a concern for the presentation of social behavior in agents and relationships that are portrayed between agents and between agents and users.

However, gaze has not been used often as a core game mechanic, apart from a few examples such as L.A. Noire [13] where the characters trying to deceive you may break eye contact. Typically, character gaze is functional, with manually-animated expressive qualities such in as Civilization VI's interstitial video clips where a sovereign announces they're angry with you and give a unique flourish [14]. Spore's [15] creatures that need to dynamically adapt to a variety of body arrangements are probably one of the most advanced examples of a game system that procedurally handles gaze; still, this is mainly functional to allow characters to survey their environment. Ultimately, since most of the animations within digital games are baked versions of animator-produced work, they are not very flexible or contextual. Therefore, to have the most potential as an effective game mechanic, gaze behavior should be proceduralized, with attention paid to its expressive and functional meanings.

To order to move in this direction by improving the quality of believable intelligent virtual agents' interactions within social settings, we examine how the important behavior of eye gaze can be improved. To do this, we created a social eye gaze model by synthesizing findings about gaze behavior and status-related behavior, drawn from both the psychological and performance art literature. These were developed into six conceptual parameters that operate on a spectrum. to assess the efficacy of the model's static parameters, we conducted an evaluation to allow us to assess it. After these studies validated the social gaze model, it was implemented in an interactive character system, which was evaluated in a lab-based study.

II. Related Work

Conversations between virtual humans and between users and such characters are an important way of promoting player engagement. On the level of physiological engagement, we know that eye gaze plays an important role in human conversations. Its role in avatar conversations has been reasonably studied, and computational models for controlling the flow of such conversations has been produced [16]. Due to its importance, and at times, central focus in animated scenes, its capabilities are well worth understanding.

Within conversations, gaze is key to regulating conversational flow, indicating attentiveness, and a variety of affective cues [17]. Argyle and Cook found that people who exhibit high amounts of mutual gaze are perceived as competent, attentive, and powerful. They also describing findings that mutual gaze between adults in Western cultures lasts an average of 1-2s. This range has been found to carry over to virtual agents [18].

Gaze behavior is also coordinated with proximity to others. Equilibrium theory [19], where people reach an equilibrium based on their interpersonal comfort. If one of the two factors is varied, the other will be changed in response. The effect of mutual gaze affecting proximity has also been observed to be similar in virtual environments [20].

For virtual characters, mutual gaze has been found to build rapport with people [21] and increase positive perceptions of affiliation [22]. A functioning gaze model has been found to motivate interest compared to a fixed-gaze [23]. Moreover, [24] examined the contributions of avatars with differing gaze patterns for conversations between remote users by comparing audio-only, random gaze avatars, inferred gaze avatars, and full video feed variations. While the random gaze model did not improve users' experience, the inferred gaze variation, which used the audio stream to determine whether a user was speaking or not, significantly outperformed both the random gaze and audio-only variations. This indicates that avatar animation should reflect the ongoing conversation. This study is also relevant because it uses a dramatic scenario involving a negotiation between aggrieved parties to provide the participants with a meaningful task to undertake.

While many advances are being made in improving intelligent virtual agents, to date, relatively few systems have been designed to directly provide them with the ability to reveal characters inner states via their gaze and/or head behavior. Some of the systems devised so far include [18] who used an animated pair of eyes to display affective signals. Queiroz and colleagues [25] devised a parametric model for automatically generating emotionally expressive gaze behavior. Lance *et al.* have presented the Gaze Warping Transformation [26], which is a means of generating expressive head and torso movements during gaze shifts based on human motion data. Their later development on GWT lead to them proposing a model of realistic, emotionally expressive gaze [27].

Busso and colleagues [28] proposed a method to generate gaze that quantizes motion-captured head poses into a finite number of clusters, and builds a Hidden Markov Model for each of these clusters based on the prosodic features of the accompanying speech. [29] proposed a parametric gaze and head movement model that is linked with the emotional state machine developed at MIRALab [30], where mood is represented with Mehrabian's Pleasure-Arousal-Dominance (PAD) Temperament Model [31]. Arousal and dominance dimensions are used to drive the parameters of the gaze and head movement models. Custom MPEG-4 Facial Animation Parameter files are then played to provide appropriate character animation.

Additionally, gaze control has been implemented in agent frameworks at various levels. Autonomic behavior is the most likely to be provided, for example with the "Eyes Alive" model

[32] for saccades and Elckerlyc's implementation of breathing and blinking [33]. Other nonverbal behaviour (NVB) tends to be generated manually by the content creator or automatically processing speech for meaningful content to cue off of.

An important precursor is the work done by Brenda Harger where she proposed using improvisational theater models to improve believable characters [34], [35]. In one study, she showed users simple animations of characters entering a room, while being able to vary the characters movement through adjusting a status parameter. By altering the parameter, participants could see the character performing the same action in different ways. Harger did not formalize a model, but this showed the potential effectiveness of demonstrating status.

Another important investigation into the development of a formalized model of NVB for portraying character attributes is [36]. They found that animators can meaningfully and reliably respond to prompts to produce character animation based on Keith Johnstone's system of Fast-Food Stanislavski.

III. The Social Eye Gaze Model

We produced a parametric model for controlling the eye-head gaze behavior of characters to portray differing levels of social status, which is shown in Table I. Cognitive models have been a mainstay in cognitive science, and can be specified at different granularities. This model is of the verbal-conceptual [37] variety, as we try to break down the components of believable gaze and assess their impact. It is a model for portraying status via gaze behavior at a medium-high level of detail, not at the minute or micro level of detail.

For example, there are existing statistical studies of saccade behavior, leading to the "Eyes Alive" model of saccades [38], which indicate how eye-only gaze changes correlate to either speaking or listening in a dyadic conversation. However, for portraying social status, behavior changes involving combinations of body parts should be considered. For this reason, the gaze model portrayed here is evaluated as a whole.

The essential components that affect the model include movement of the important qualifiers such as use of personal space and overt social signaling. The model is based on a number of psychological theories affecting the relationship between these qualifiers and the behavior performance traits that vary depending on personality type. After surveying the literature to compile the most relevant theories, we extracted the descriptions of NVB in dyadic conversations. We then grouped these into the aspects of gaze behavior that were affected. The resulting parameters are labeled following the naming convention for these phenomena established in [39].

The model contains six important components whose variation has been found to signify differences in social status. Table I provides a breakdown of the parameterized behavior that affects the general pattern described above along several continua. Movement toward an end of a continuum occurs simultaneously in the same direction along the other continua, although each parameter will have differently sized differences between gradations. *Length of eye contact* refers to the period of time during which mutual eye gaze is occurring, measured in seconds. *Length of gazes* describes how long gazes toward the opposite person in the dyad last for, measured in seconds. *Movement velocity* refers to the speed of the eyes, head, and neck during shifts in gaze, measured in degrees per second. *Head posture* describes the overall inclination of the head. *Equilibrium response* indicates the immediate response made when the opposite person in the dyad moves into a closer personal reaction bubble. *Posture* indicates a general stance affecting the entire torso.

Table I
The parameters of the social eye gaze model for portraying status through variations in gaze

Parameter	Low Status Behavior	High Status Behavior
Length of eye contact	Shorter	Longer
Length of gazes	Shorter	Longer
Movement velocity	Fast	Slow
Head posture	Bowed	Raised
Equilibrium response	Look away sooner	Stare longer
Posture	Lean away	Lean toward

IV. Evaluation of the Social Eye Gaze Model Parameters

We wanted to determine whether the listed parameters produce behaviour that functions as major drivers of audience perception. Furthermore, while we used the baseline established for humans in the psychological literature, we wanted to find workable ranges of values for these parameters. To do so, we first implemented a scenario within SmartBody that would facilitate the comparison of behavioral differences. The videos of these scenarios were then shown to participants using Amazon's Mechanical Turk (MTurk) platform to compare the conditions and find effective values for the parameters. MTurk has been generally found to be a reliable mechanism for conducting experiments [40].

In the empirical evaluations, we test the hypothesis that an animated character whose gaze movements are faster, whose fixations are shorter, and whose perceived length of eye contact is shorter, will be perceived as lower status than one whose gaze movements are slower, whose fixations are longer, and whose perceived length of eye contact is longer. We do so by comparing the participants' rating of the character's status.

A. Method

To demonstrate the feasibility of our model, we used the SmartBody procedural animation framework [41] to animate a short scene from Keith Johnstone's book on improvisational theater [8]. The scene for this study is set in an office environment with a corporate officer who fires an employee with a criminal record. We chose to use this scenario because it has been successfully used to evaluate the NVB of characters [36], [42], and it explores a situation with dramatic behaviour where social status plays a meaningful role. It provides a simple scene that constrains a number of factors including the length of the exchange, the number of actions that occur, and the personality traits and emotions on display. In addition, based on findings in psychological studies, 30 seconds of

exposure to people is long enough to make an impression about their personality [43], without tiring participants. This scenario was implemented using BML to generate the dialog, as well as a minimal set of accompanying gestures (e.g. head shakes for negative statements, deictic hand waves to refer to the other person) to avoid presenting unnaturally still avatars. These were designed to be emotionally neutral and were not varied between conditions.

1) Experimental Design: The scenario was produced to support two conditions, with a corporate officer displaying either high status or low status behavior as they fired an employee. This was achieved by varying only behavior associated with the following three model parameters: length of eye contact, length of fixations, and movement velocity. The scenario was implemented in the SmartBody environment follow the scenario's script.

Four different evaluations were performed. For each of them, one human intelligence task (HIT) with 50 assignments was loaded onto MTurk; thus, 50 unique workers completed both trials, for a total of 200 participants. We restricted the participants to be those with US accounts. Overall, the average age of the 200 participants was 35.0 (SD = 11.0), with 57% reporting their gender as male; the remainder reported female.

In the first evaluation, we used audio recorded by local actors. The corporate officer was represented with a masculine character, and the employee by a feminine character.

In the second evaluation, we replaced the actors' voices with TTS voices. The computer-generated voices were SAPI-compliant male and female voices from Cereproc.

In the third evaluation, we rotated the camera 180° from the first evaluation's setup to focus on the employee character, whose NVB was also generated according to the social eye gaze model. The actors' voices were retained.

To assess to which degree gender plays a role in the perception of status, we tested a forth factor, using a female character as the corporate officer. In this pair of videos, the camera framed the Corporate Officer character as in the first evaluation, while the scenario remained the same. The characters' voices used the same TTS voices.

We used the Ten-Item Personality Inventory Measure (TIPI) [44] which is frequently used for measuring the "Big Five" personality traits (openness, conscientiousness, extraversion, agreeableness, and emotional stability). TIPI has been used successfully in assessing virtual characters before, e.g. [18].

2) Procedure: The experiment was registered on the MTurk website, and workers found it by browsing for HITs. They viewed the experiment "ad", and this contained the task instructions and informed consent. If they accepted the HIT, they then saw the system capability verification screen. Next, they provided some basic demographic information (age, gender). Then, for each condition, they viewed the video of the scenario and then rated the character. After each video, participants completed the TIPI (10 questions), which uses a 7-point Likert scale response. They were then prompted: "Given that *social status* is a person's standing or importance in relation to other people, please circle the character's apparent social status". The status question was answered using a 5-point Likert scale, rated from 1 (Very Submissive) to 5 (Very Dominant).

The experiment was presented to workers on MTurk as an HTML webpage with task flow controlled by JavaScript code running locally in each worker's web browser. Prior to the experiment workers verified their system functionality by listening to a brief audio track and transcribing the text. Trials were constructed by pairing videos of the scenarios in the different conditions of High and Low Status. The participant rated the character's personality and status after each one. Randomizing the order of the videos counterbalanced the conditions.

B. Results

In our statistical analysis we compared the change in status rating across the different experimental conditions.

A univariate repeated measures analysis of variance was conducted to compare the effect of social gaze model condition on status. Results indicated there was a significant difference between the High Status condition (M = 3.64, SD = 1.16) and the Low Status condition (M = 3.45, SD = 1.24), $F(1, 196) = 5.93$, $p = .016$. No statistically significant interactions between the other conditions (character, voice) were present in this analysis. This finding supports our hypothesis and suggests that the differing gaze behavior produced by the model does have an effect on the perceived social status of animated characters.

V. Interactive Evaluating the Dynamic Social Gaze Model

This section presents a study that investigates how to send social signals related to status by varying a virtual human's eye gaze and how people evaluate virtual humans engaging them in social behavior. We integrated the social eye gaze model into a behavior control module for a character animated with the SmartBody procedural animation system. By combining this with a chatbot-driven dialog system, we created a job interview simulator with two different variations on the character's eye gaze. Participants then practiced interviewing the characters while wearing an eye tracker, which allowed the character's behavior to vary based on mutual eye gaze. Again, they reported that the varying gaze model changed their impression of the character's social status, providing further support for our social gaze model.

A. Method

1) System Design: The system, as shown in Figure 1, facilitates gathering input from a participant, and simultaneously controlling a SmartBody embodied conversational agent (ECA) based on that input. The two inputs are eye tracking data, which is monitored for mutual gaze with the ECA, and voice input, which is used to generate dialog responses. All scripts and libraries were programmed in Python.

The main sensor used is an eye tracking headset from Pupil Labs [45], which coordinates a gaze camera and a world camera to determine where the participant is looking. We used

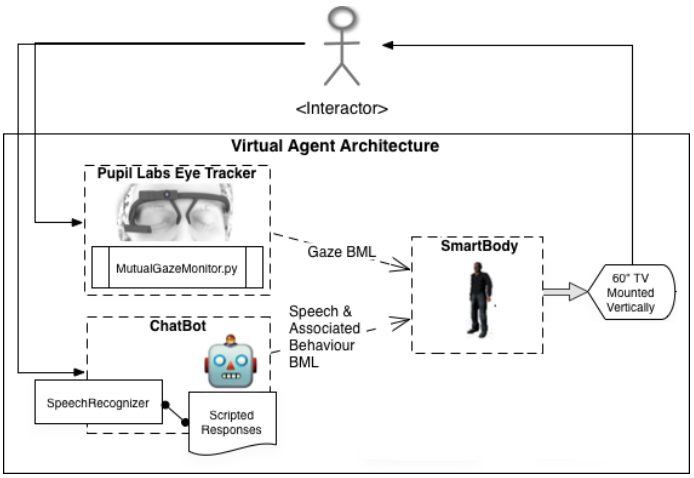


Figure 1. Diagram of the virtual agent architecture for eye tracking study.

the world camera's medium setting of 1280x720 to capture 60fps. The gaze camera captured 640x480 at 120fps, using IR illumination. Specifically, we defined a virtual region on the screen surrounding the character's eyes. These "virtual surfaces" can be seen in Figure 2, registered relative to fiducial markers. Doing so allowed Pupil Capture to record and stream information about the participants' gazes on the specified surface, generating surface visibility reports and gaze counts. This was found to be a very useful way to stream real-time information about gaze on a user-defined area of interest within an arbitrary software application.

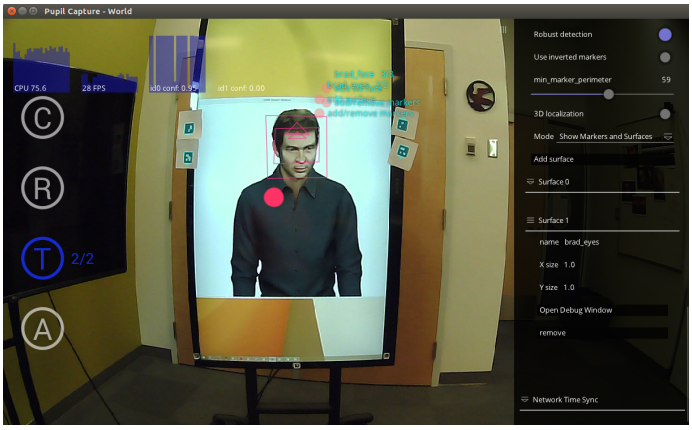


Figure 2. Sample eye tracking surfaces as seen in Pupil Capture.

To provide dialog for the system, we used a custom chat bot [46], [47], which uses the Google Speech API to obtain audio from a microphone, and then relies on a probabilistic match system to provide the best response to a given prompt [48]. We provided the bot with dialog relating to a hiring scenario. The chat bot uses StompSender, which is a Python class that can send control messages to SmartBody via Apache ActiveMQ using the STOMP protocol. The chat bot produced vocal output using computer-generated voices (TTS) that were SAPI-compliant male and female voices. TTS voices were used since being able to rely on procedural speech is important to the overall design goals of flexibility and portability in a virtual human system.

Finally, the mutual gaze behavior is managed by custom Python scripts. Whenever contact is made with the virtual character's eye region, the script determines how long the character will sustain mutual gaze for. If the contact is maintained, then eventually the script will direct the character to avert their eye-head gaze. We implemented two different versions of this behavior, to correspond to the experimental conditions. In the High Status condition, the wait time is based on a random number chosen from a Gaussian distribution with M = 6.0s and SD = 1.0s. In the Low Status condition, the wait time is based on a random number chosen from a Gaussian distribution with M = 4.0s and SD = 0.5s. These numbers were chosen to produce a range of times similar to those in the psychological literature and tweaked with adjustments based on early pilot studies. The other aspects of the gaze behavior were timed to provide the maximum realistic separation, based on the spectrum of parameters in the social eye gaze model.

2) Procedure: All experimental sessions were conducted individually. The experimenter greeted each participant, and explained the purpose of the study and the consent form. Once the participant signed the consent form, the participant completed a demographic data form and reviewed a set of instructions for the interaction. To interact with the system, the participant sat at a desk and wore the eye tracking headset. Then, a brief calibration routine was conducted to ensure the eye tracking was functioning properly. The participant then interacted with the character in a simulated hiring scenario, by asking the virtual character questions. On the desk was placed a sheet of potential questions, a microphone, and a mouse that could be clicked to allow the system to receive voice input.

After each condition, just as in the previous step, the avatar's personality was assessed using the Ten-Item Personality Inventory (TIPI), and a rating of the character's apparent social status on a Likert scale from 1 (Very Submissive) to 5 (Very Dominant). The participants were also asked to describe their main reason for the status rating. They were also asked to rate the character's competence similarly. They then repeated the process with the other condition. The study took about 30 minutes to complete.

The participants were recruited from undergraduate and graduate students. The undergraduate participants received nominal credit from their course instructors for their participation. The remainder of the participants were volunteers. 35 participants were recruited for this study. Typically, eye tracking usability-style studies recruit between 6–30 participants, although oversampling is recommended due to the potential for technical issues [49]. However, one participant reported that they had difficulty paying attention to the task, and subsequent review of their eye tracking data showed they only looked at the animated character's head region seven times during a session, which was well below the average number of such gazes across participants (M=2440). As perceiving the character's activity was the major focus of the study, and a

Figure 3. The visual portrayal of the two versions of the character. (left) Brian; (right) Brad.

prerequisite for making judgments about it, this participant's data was therefore excluded from the subsequent descriptions and analyses.

Of the 34 participants, 24 of them were aged 20–24. 18 were undergrad students. 11 indicated their gender was male, and 24 indicated their gender was female. Regarding their experience with video games, 19 of them chose responses that indicated they didn't play games or did occasionally, but wouldn't identify with the term gamer. 15 of them indicated they were either casual or hardcore gamers. The average number of times they interviewed someone else in a work setting previously was 2 (SD = 3), while the average number of times they were interviewed for work was 6 (SD = 6). Most had not interviewed someone else, but had been interviewed for work purposes. The observations of this study are therefore expected to generalize to young adults with some higher education, and minor work and video game experience.

The participants engaged in a scripted interaction with our virtual human character. They were given a list of questions they could ask in a simulated hiring interview. There were two versions of the character, with its gaze behavior adapted to represent a person of either high or low status. Participants interacted with both versions of the characters, in a within-subjects design. The variations of the character were assigned a differently colored shirt and name ("Brian" wore a dark gray shirt and "Brad" wore a blue shirt, as shown in Figure 3), and the study was counter-balanced by presenting the two conditions in a random order.

B. Results

The statistical analysis of the study, wherein participants interacted with an interactive character compared the effects of the two different conditions. Order of first condition did not have a statistically significant effect on the results; neither did participant gender and age range, which was the case even when assessing virtual characters and gender stereotyping [50].

We performed a paired-sample t-test comparing the social status rating results for the character in both conditions. There is a significant difference in the scores for the High Status condition (M = 3.18, SD = 0.72) and the Low Status condition (M = 2.79, SD = 0.98); $t(33) = 2.03$, $p = .046$. These results suggest that the differing gaze behavior really does have an effect on the perceived social status of interactive animated characters. Cohen's d = 0.45, which is in the range of a Medium effect. This finding supports our hypothesis.

We performed a paired-sample t-test comparing the competence rating results for the character in both conditions. There was no significant difference in the scores for the High Status condition (M = 3.85, SD = 0.93) and the Low Status condition (M = 3.74, SD = 1.11); $t(33) = 2.03$, $p = .54$.

Paired-sample t-tests comparing the five personality factors from each condition found no significant difference. This implies that the use of the model did not alter the perceived personality of the character in this evaluation.

VI. Discussion

A. The Parameterized Gaze Model

This study determined that a specific combination of gaze behaviors could be used to provide virtual humans with the ability to communicate social status. Implementing the social gaze model for these static videos allowed us to develop the necessary BML commands and timing values that could be used when the model is implemented within an interactive system, and so this effectively allowed us to prototype the required behavior for the interactive study in Section V. Finding empirical support for the effects of the social gaze model's parameters also gave us confidence it would be effective and worthwhile evaluating further.

While participants mentioned receiving an emotional impression of the character in the open-ended comments, such as appearing anxious and uncertain, the assessment of social status was still effective. A few comments did make direct reference to the characters' eye movements. The fact that they brought up emotional impressions rather than the gaze behavior actually helps show that social status is perceived without being overwhelming. Overall, participants reported that the characters' roles made the biggest impression on their rating of social status, with NVB as the second-most frequent factor. This shows the importance of the narrative context for indicating to players why character actions occur.

B. Interactive Study

In this study, participants interacted with an ECA that used the social eye gaze model's fundamental values to control its dynamic behavior, while adding in the parameter of length of eye contact through its ability to respond to mutual eye gaze. Since this study showed additional support for our hypothesis, it serves as additional validation of the model. Importantly, it shows support for the novel element added in this study: varying response according to mutual gaze.

This finding is in contrast to at least one previous finding, wherein a character implemented in Elckerlyc was used to mediate communication between two people [51]. There, participants saw each other as an avatar while conversing, and its gaze was delayed to see if it thereby mimicked high status behavior and portrayed dominance. The researchers weren't

able to support that hypothesis. In that case, participants knew they were communicating to another person through the avatar, so it is possible that participants cognitively assessed it differently than a communicating character, or that the selected delay didn't operationalize into a signal of social status. Additionally, a study of gaze in interpersonal interviews found that gaze aversions had strong negative connotations such as perceptions of lowered credibility and attraction [52], whereas nearly constant levels of high gaze are not significantly more favorable than normal levels. Thus, it may be that gaze behavior sends signals of low status more readily than the inverse. This was supported in the comments about the Low Status condition in our study. Participants associated the NVB they saw with low status, while the high status behaviour was often viewed in neutral terms, and their comments switched to focus on the content in that condition.

The assessment of competence in this study was an attempt to determine if there was any connection between the gaze model and participations' perception of this characteristic, following prior findings that interviewees using normative to nearly constant gaze patterns were seen as more competent [52]. We did not find that, and indeed most participants commented on the chatbot's dialogue options and ability to respond to questions when asked about their reasoning. It seems that task performance is a priority in this context.

It is also worth noting that this study examined a different scenario than those in Section IV. Those videos examined a dramatic scenario taken from improv acting related to a firing incident. In this case, participants participated in a practice interview session. However, it seemed important to evaluate the gaze model parameters in an interactive setting. The hiring scenario is a more constrained and formulaic experience. Still, training simulators represent an important venue for ECAs [53], even in sophisticated contexts such as training psychologists to do interviews [54]. Still, both settings represent a range of experience that is important to equip virtual humans for.

VII. Conclusion

In this paper we propose a verbal-conceptual procedural model that provides a correspondence between the dramatic quality of social status and six parameters affecting eye gaze behavior. The studies documented in this paper provide evidence for the efficacy of its parameters, through interaction with both static and interactive representations of virtual humans.

These studies show the importance of gaze to the social interaction of interactive virtual agents, particularly during conversations. Consisting of coordinated motion between the torso, head, and eyes, gaze provides a broad range of abilities including general vision, task-monitoring, emotional display, and even subtler social signals in terms of the social status of participants. Status was identified as a valuable realistic and dramatic component of communication, worth investigating for its contribution to believability. As we create intelligent virtual agents that are intended to be dramatically believable, the concern for meeting viewer expectations surrounding communication as well as providing the ability for subtler contrasting messages leads us to consider ways of improving the gaze capability of virtual avatars.

As mentioned, [55] proposes a means of guiding participants by creating status vacuums, based on the principles of interactive theater. In that case, the believability level of the agent is important for building user empathy and interest in the dramatic outcome. If social status is a reliable way of changing participant behavior, then this model of believable social gaze could help improve the portrayal of characters involved in similar kinds of scenarios, as well as contribute to a new way of directing player actions during dramatic scenes.

Social status is a powerful concept that has the potential to produce believable behavior in virtual humans. In improv theater, status refers to power differences in the relationship between two characters; additionally, the most relevant differences are seen in actions taken, rather than overt differences such as apparent wealth and rank. Since Johnstone believes that the most interesting scenes arise out of status changes, it is essential that it is possible to portray characters of different status and have means for changing status. Our social eye gaze model will enable virtual humans with a reliable way to communicate status and thus play a part in diverse dramatic situations. This could increase the number of contexts where they can be used, including more genres of digital games, especially those incorporating social simulations to provide lifelike activity.

Acknowledgment

We would like to thank our reviewers for their very helpful feedback.

References

[1] M. Wooldridge and N. R. Jennings, "Intelligent agents: Theory and practice," *Knowledge Engineering Review*, vol. 10, no. 2, pp. 115–152, 1995.

[2] A. S. Rao and M. P. Georgeff, "Modeling rational agents within a bdi-architecture," *Readings in agents*, pp. 317–328, 1997.

[3] M. Mateas and A. Stern, *Procedural authorship: A case-study of the interactive drama Faade*. IT University of Copenhagen, Dec 2005.

[4] W. R. Swartout, J. Gratch, R. W. Hill Jr, E. Hovy, S. Marsella, J. Rickel, and D. Traum, "Toward virtual humans," *AI Magazine*, vol. 27, no. 2, p. 96, 2006.

[5] R. Aylett, S. Louchart, J. Dias, A. Paiva, and M. Vala, *Fearnot!: an experiment in emergent narrative*, ser. Lecture Notes in Computer Science. Springer-Verlag, 2005, vol. 3661/2005, pp. 305–316.

[6] J. Cassell, H. Vilhjlmsson, and T. Bickmore, *BEAT: the Behavior Expression Animation Toolkit*. ACM, 2001, pp. 477–486.

[7] A. Vinciarelli, M. Pantic, H. Bourlard, and A. Pentland, *Social signals, their function, and automatic analysis: a survey*, ser. ICMI 08. ACM, 2008, pp. 61–68, aCM ID: 1452405.

[8] K. Johnstone, *Impro: Improvisation and the Theatre*. Methuen Drama, 1981.

[9] V. Spolin, *Improvisation for the theater: A handbook of teaching and directing techniques*. Northwestern University Press, 1983.

[10] A. Kendon, "Some functions of gaze direction in social interaction," *Acta Psychologica*, vol. 26, no. 1, pp. 22–63, 1967.

[11] M. Shimada, T. Minato, S. Itakura, and H. Ishiguro, "Evaluation of Android Using Unconscious Recognition," in *2006 6th IEEE-RAS International Conference on Humanoid Robots*. IEEE, dec 2006, pp. 157–162. [Online]. Available: http://ieeexplore.ieee.org/document/4115595/

[12] M. Mori, "The uncanny valley," *Energy*, vol. 7, no. 4, pp. 33–35, 1970.

[13] Team Bondi, "L.A. Noire," 2011.

[14] Firaxis Games, "Civilization VI," 2016.

[15] Maxis, "Spore," 2008.

[16] C. Peters, C. Pelachaud, E. Bevacqua, M. Mancini, and I. Poggi, "A Model of Attention and Interest Using Gaze Behavior," in *Intelligent Virtual Agents*, ser. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, Sep. 2005, pp. 229–240. [Online]. Available: https://link-springer-com.proxy.lib.sfu.ca/chapter/10.1007/11550617_20

[17] M. Argyle and M. Cook, *Gaze and mutual gaze*. Cambridge University Press, 1976.

[18] A. Fukayama, T. Ohno, N. Mukawa, M. Sawaki, and N. Hagita, *Messages Embedded in Gaze of Interface Agents — Impression Management with Agent's Gaze*, ser. CHI '02. New York, NY, USA: ACM, 2002, pp. 41–48. [Online]. Available: http://doi.acm.org/10.1145/503376.503385

[19] M. Argyle and J. Dean, "Eye contact, distance, and affilliation," *Sociometry*, vol. 28, pp. 289–304, 1965.

[20] J. Bailenson, J. Blascovich, A. Beall, and J. Loomis, "Interpersonal distance in immersive virtual environments," *Personality and Social Psychology Bulletin*, vol. 29, no. 7, p. 819, 2003.

[21] N. Wang and J. Gratch, *Dont just stare at me!* ACM, 2010, pp. 1241–1250. [Online]. Available: http://dl.acm.org/citation.cfm?id=1753513

[22] S. Andrist, T. Pejsa, B. Mutlu, and M. Gleicher, *Designing effective gaze mechanisms for virtual agents*. ACM, 2012, pp. 705–714. [Online]. Available: http://dl.acm.org/citation.cfm?id=2207777

[23] A. Colburn, M. F. Cohen, and S. Drucker, "The role of eye gaze in avatar mediated conversational interfaces," *Sketches and Applications, Siggraph00*, 2000. [Online]. Available: https://pdfs.semanticscholar.org/2e9a/7e7654d7a1f9c92598f2cc8711dc19addd3c.pdf

[24] M. Garau, M. Slater, S. Bee, and M. A. Sasse, *The impact of eye gaze on communication using humanoid avatars*. ACM, 2001, pp. 309–316. [Online]. Available: http://dl.acm.org/citation.cfm?id=365121

[25] R. B. Queiroz, L. M. Barros, and S. R. Musse, "Providing expressive gaze to virtual animated characters in interactive applications," *Computers in Entertainment (CIE)*, vol. 6, no. 3, p. 41, 2008.

[26] B. Lance and S. C. Marsella, "Emotionally expressive head and body movement during gaze shifts," in *Intelligent Virtual Agents*, C. Pelachaud, J.-C. Martin, E. André, G. Chollet, K. Karpouzis, and D. Pelé, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 72–85.

[27] B. J. Lance and S. C. Marsella, "A model of gaze for the purpose of emotional expression in virtual embodied agents," in *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 1*, ser. AAMAS '08. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2008, pp. 199–206. [Online]. Available: http://dl.acm.org/citation.cfm?id=1402383.1402415

[28] C. Busso, Z. Deng, U. Neumann, and S. Narayanan, *Learning expressive human-like head motion sequences from speech*. Springer, 2008, pp. 113–131. [Online]. Available: https://link.springer.com/content/pdf/10.1007/978-1-84628-907-1_6.pdf

[29] C. Cig, Z. Kasap, A. Egges, and N. Magnenat-Thalmann, *Realistic emotional gaze and head behavior generation based on arousal and dominance factors*, ser. Lecture Notes in Computer Science. Springer-Verlag, 2010, pp. 278–289. [Online]. Available: http://link.springer.com/content/pdf/10.1007/978-3-642-16958-8.pdf#page=289

[30] Z. Kasap, M. B. Moussa, P. Chaudhuri, and N. Magnenat-Thalmann, "Making them rememberemotional virtual characters with memory," *IEEE Computer Graphics and Applications*, vol. 29, no. 2, pp. 20–29, 2009.

[31] A. Mehrabian, "Analysis of the big-five personality factors in terms of the pad temperament model," *Australian Journal of Psychology*, vol. 48, no. 2, pp. 86–92, 1996.

[32] S. Kopp, B. Krenn, S. Marsella, A. Marshall, C. Pelachaud, H. Pirker, K. Thrisson, and H. Vilhjlmsson, *Towards a common framework for multimodal generation: The behavior markup language*, ser. LNCS (LNAI). Springer, 2006, vol. 4133, pp. 205–217.

[33] H. Van Welbergen, D. Reidsma, Z. Ruttkay, and J. Zwiers, "Elckerlyc-a bml realizer for continuous, multimodal interaction with a virtual human," *Journal on Multimodal User Interfaces*, vol. 3, no. 4, pp. 271–284, 2010.

[34] B. Harger, *Workshop on Improvisational Acting*, 2007.

[35] ——, *Entertaining AI: Using Rules from Improvisational Acting to Create Unscripted Emotional Impact*, 2008.

[36] M. Seif El-Nasr and H. Wei, *Exploring Non-verbal Behavior Models for Believable Characters*. Springer, Nov 2008, vol. 5334, pp. 71–82.

[37] R. Sun, *Introduction to computational cognitive modeling*. Cambridge University Press, 2008, pp. 3–19. [Online]. Available: http://cindy.informatik.uni-bremen.de/cosy/teaching/CM_2011/intro/sun_08.pdf

[38] S. P. Lee, J. B. Badler, and N. I. Badler, *Eyes alive*. ACM Press, 2002, vol. 21, pp. 637–644. [Online]. Available: http://portal.acm.org/citation.cfm?doid=566570.566629

[39] B. J. Lance and S. C. Marsella, "Glances, glares, and glowering: how should a virtual human express emotion through gaze?" *Autonomous Agents and Multi-Agent Systems*, vol. 20, no. 1, pp. 50–69, May 2009.

[40] M. J. Crump, J. V. McDonnell, and T. M. Gureckis, "Evaluating amazons mechanical turk as a tool for experimental behavioral research," *PloS one*, vol. 8, no. 3, p. e57410, 2013.

[41] M. Thiebaux, S. Marsella, A. N. Marshall, and M. Kallmann, "Smartbody: Behavior realization for embodied conversational agents," in *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 1*, ser. AAMAS '08. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2008, pp. 151–158. [Online]. Available: http://dl.acm.org/citation.cfm?id=1402383.1402409

[42] M. Nixon, "Enhancing believability: Evaluating the application of delsartes aesthetic system to the design of virtual humans," Master's thesis, Simon Fraser University, Oct 2009.

[43] N. Ambady and R. Rosenthal, "Thin slices of expressive behavior as predictors of interpersonal consequences: A meta-analysis." *Psychological Bulletin*, vol. 111, no. 2, pp. 256–274, 1992.

[44] S. D. Gosling, P. J. Rentfrow, and W. B. Swann, "A very brief measure of the big-five personality domains," *Journal of Research in Personality*, vol. 37, no. 6, pp. 504–528, 2003.

[45] M. Kassner, W. Patera, and A. Bulling, *Pupil: an open source platform for pervasive eye tracking and mobile gaze-based interaction*. ACM, 2014, pp. 1151–1160. [Online]. Available: http://dl.acm.org/citation.cfm?id=2641695

[46] S. S. Feldman, O. N. Yalcin, and S. DiPaola, "Engagement with artificial intelligence through natural interaction models," in *Proceedings of the Conference on Electronic Visualisation and the Arts*, ser. EVA '17. Swindon, UK: BCS Learning & Development Ltd., 2017, pp. 296–303. [Online]. Available: https://doi.org/10.14236/ewic/EVA2017.60

[47] O. N. Yalcin and S. DiPaola, "Virtual AI agent for Adults with Mental Health Issues," in *Stanford Workshop on VR and Behavioral Change, Poster*, Palo Alto, California, 2017.

[48] A. Leuski, R. Patel, D. Traum, and B. Kennedy, *Building effective question answering characters*. Association for Computational Linguistics, 2009, pp. 18–27. [Online]. Available: http://dl.acm.org/citation.cfm?id=1654600

[49] J. H. Goldberg and A. M. Wichansky, "Eye tracking in usability evaluation: a practitioners guide," in *The Mind's Eyes: Cognitive and Applied Aspects of Eye Movements*, J. Hyn, R. Radach, and H. Deubel, Eds. Oxford: Elsevier Science, 2002.

[50] J. Fox and J. N. Bailenson, "Virtual virgins and vamps: The effects of exposure to female characters sexualized appearance and gaze in an immersive virtual environment," *Sex roles*, vol. 61, no. 3–4, pp. 147–157, 2009.

[51] B. Gennep, "In the eye of the wizard: effects of (mutual) gaze on an avatar mediated conversation," Ph.D. dissertation, University of Twente, 2013. [Online]. Available: http://essay.utwente.nl/64986/

[52] J. K. Burgoon, D. A. Coker, and R. A. Coker, "Communicative effects of gaze behavior," *Human Communication Research*, vol. 12, no. 4, pp. 495–524, 1986.

[53] J. Rickel, *Intelligent virtual agents for education and training: Opportunities and challenges*. Springer, 2001, pp. 15–22. [Online]. Available: http://link.springer.com/content/pdf/10.1007/3-540-44812-8.pdf#page=24

[54] P. Kenny, A. A. Rizzo, T. D. Parsons, J. Gratch, and W. Swartout, "A virtual human agent for training novice therapist clinical interviewing skills," *Annual Review of CyberTherapy and Telemedicine*, vol. 5, pp. 77–83, 2007.

[55] J. Zhu, K. Ingraham, and J. Moshell, *Back-Leading through Character Status in Interactive Storytelling*, ser. Lecture Notes in Computer Science. Springer, 2011, vol. 7069/2011, pp. 31–36.

Evolutionary Multi-objective Optimization of Real-Time Strategy Micro

Rahul Dubey, Joseph Ghantous, Sushil Louis and Siming Liu
Computer Science and Engineering
University of Nevada
Reno, USA
rdubey018@nevada.unr.edu, joseph.ghantous@gmail.com, sushil@cse.unr.edu, simingl@cse.unr.edu

***Abstract*—We investigate an evolutionary multi-objective approach to generating micro for real-time strategy games. Good micro helps a player win skirmishes and is one of the keys to developing better real-time strategy game play. In prior work, the same multi-objective approach of maximizing damage done while minimizing damage received was used to evolve micro for a group of ranged units versus a group of melee units. We extend this work to consider groups composed from two types of units. Specifically, this paper uses evolutionary multi-objective optimization to generate micro for one group composed from both ranged and melee units versus another group of ranged and melee units. Our micro behavior representation uses influence maps to represent enemy spatial information and potential fields generated from distance, health, and weapons cool down to guide unit movement. Experimental results indicate that our multi-objective approach leads to a Pareto front of diverse high-quality micro encapsulating multiple possible tactics. This range of micro provided by the Pareto front enables a human or AI player to trade-off among short term tactics that better suit the player's longer term strategy - for example, choosing to minimize friendly unit damage at the cost of only lightly damaging the enemy versus maximizing damage to the enemy units at the cost of increased damage to friendly units. We believe that our results indicate the usefulness of potential fields as a representation, and of evolutionary multi-objective optimization as an approach, for generating micro.**

***Index Terms*—NSGA-II, influence maps, potential fields, game AI**

I. Introduction

Real-Time Strategy games provide difficult challenges for computational intelligence researchers seeking to build artificially intelligent opponents and teammates for such games. In these games, players find and consume resources to build an economy, build an army to defeat an opponent in a series of skirmishes usually culminating in a large decisive battle. Good RTS game play embodies near-optimal sequential decision making in an uncertain environment under resource and time constraints against a deceptive, dynamic, and adaptive opponent (when playing against good players). Researchers have thus begun focusing on real-time strategy games as a new frontier for computational and artificial intelligence research in games [1].

RTS game play involves both long-term strategic planning and shorter term tactical and reactive actions. The long-term planning and decision making, often called macromanagement, or just macro for short, can be contrasted with the quick but precise and careful control of game units in order to maximize unit effectiveness on the battlefield. This short-term control and decision making is often called micromanagement, or just micro and good micro can win skirmishes even when a player has fewer units. This paper focuses on evolving micro for groups of units of different types.

Although much diverse work has been done on generating micro for RTS games, our work differs in two aspects. First, we use evolutionary multi-objective optimization to tradeoff two objectives: damage done versus damage received. Second, we represent unit behavior using multiple potential fields and an influence map whose parameters evolve to generate micro for groups composed from two types of units. Potential fields of the form cx^e where x can be distance, health, or weapons cooldown determine unit movement. Influence maps that give high values to map locations with more opponent units specify the location to move towards or to attack. This paper extends earlier work that used the same representation and Evolutionary Multi-Objective Optimization (EMOO) approach in evolving micro for one type of melee unit versus one type of ranged unit [2]. In this work, we use our own implementation of the NSGA-II algorithms by Deb [3].

Our results indicate that we can evolve micro for a group of ranged and melee units versus a group of the same number and types of ranged and melee units. The evolved micro performs well against hand selected opponents under a variety of conditions. Without explicit representation, we see the emergence of kiting behavior for the ranged units, fleeing behavior for the melee units, and strong melee units screening for the relatively weak ranged units. Kiting refers to attack and flee behavior. Fleeing refers to running away. The pareto front of evolved solutions contains a variety of tactics suitable for a variety of roles in the broader strategic situation in a particular game. For example, the GA evolves micro that maximizes damage to opponent units while also receiving significant damage, more balanced micro that deals and receives approximately equal amounts of damage, as well as micro that deal little damage but also receives little damage. In the broader picture, this enables a human or AI player to choose the appropriate micro for the current strategic situation. For example, a player may choose micro that prefers to reduce damage by harassing because it will tend to draw away opponent units from the main force or occupy existing opponent units at a distant

978-1-5386-4359-4/18/$31.00 ©2018 IEEE

location. We believe these results indicate the potential of a multi-objective approach for evolving high performance micro and to the potential for a potential fields representation of tactical behavior.

The remainder of this paper is organized as follows. Section II discusses related work in RTS AI research and common approaches to evolve the micro behavior of units. Section III describes our 3D simulation platform, FastEcslent. Section IV introduces the pure potential fields and influence maps that govern micro in simulated skirmishes. This section also describes the NSGA-II algorithm used to evolve the micro behavior. Section V presents results and discussion. Finally, the last section draws conclusions from our results and discusses future work.

II. Related Work

RTS AI work is popular in both industry and academia. Industry RTS AI developers are more focused on entertainment while academic RTS AI research focuses on learning or reasoning techniques for winning. RTS game play involves both macro-management; long term strategic planning and micro-management; short term tactical actions. Generating optimal build orders that produce a needed mix of unit types falls under the category of long term planning, that is, macro. Since macro produces the units used for micro, we start with work in macro for RTS games. Ballinger evolved robust build orders in WaterCraft [4]. Gmeiner proposed an evolutionary approach for generating optimal build orders [5]. Kostler evolve strategies for either producing units of more types or producing more number of units as quickly as possible [6]. Once we have good macro producing a given a set of units, micro deals with controlling these units and there is strong research interest in producing effective group behavior (good micro) in skirmishes since good micro can often turn the tide in close battles. Liu used case-injected genetic algorithm to generate high quality micro [7]. Churchill presented a fast search method based on alpha-beta considering duration (ABCD) algorithm for tactical battles in RTS games [8]. Again, Liu investigated hill climbers and canonical GAs to evolve micro behaviors in RTS games showing that genetic algorithms were generally better in finding robust, high performance micro [9]. Louis and Liu evolved effective micro behavior based on influence maps and potential fields in RTS games [10]. Our paper extends the work in [2] and represents micro based on influence maps and potential fields for spatial reasoning and unit movement.

In physics, a potential field is usually a distance dependent vector field generated by a force. The concept of artificial potential field was first introduced by Khatib for robot navigation and later this concept was found useful in guiding movement in games [11]. An influence map structures the world into a 2D or 3D grid and assigns a value to each grid element or cell. Liu compares two different micro representations and the result indicate that even with less domain knowledge the potential fields based representation can evolve a reliable, high quality micro in a three dimensional RTS game [12]. Schmitt worked on evolutionary competitive approach to evolve micro using potential fields based micro representation and results shows that their approach can evolve complex units movement during skirmish [13].

Early work used influence maps for spatial reasoning to evolve a LagoonCraft RTS game player [14]. Sweetser presented an agent which uses cellular automata and influence maps for decision-making in 3D game environment called EmerGEnt [15]. Bergsma proposed a game AI architecture which use influence maps for a turn based strategy game [16]. Preuss investigated an evolutionary approach to improve unit movement based on flocking and influence map in the RTS game Glest [17]. Uriarte presented an approach to perform kiting behavior using Influence Maps in multi-agent game environment called Nova [18].

Cooperation and coordination in multi-agent systems, was the focal point of many studies [19], [20], [21], [22]. Reynolds early work explores an approach to simulate bird flocking by creating a distributed behavioral model that results in artificial agent behavior much like natural flocking [23]. Similarly Chuang studied controlling large flocks of unmanned vehicles using pairwise potentials [24].

Within the games community, Yannakakis [25] evolved opponent behaviors while Doherty [26] evolved tactical team behavior for teams of agents. Avery used an evolutionary computing algorithm to generate influence map parameters that led to effective group tactics for teams of entities against a fixed opponent [27], [28]. We define potential fields and influence maps in more detail later in the paper. This paper extends Louis' [2] work in dealing with micro for heterogeneous groups of units.

To run our experiments we created a simulation model similar to StarCraft-II called FastEcslent, our open source, 3D, modular, RTS game environment. The next section introduces this simulation environment in more detail.

III. Simulation Environment

With the release of the StarCraft-II API, StarCraft: Brood War API (BWAPI) and numerous tournaments such as Open Real-Time Strategy Game AI Competition, the Artificial Intelligence and Interactive Digital Entertainment StarCraft AI Competition, and the Computational Intelligence and Games StarCraft RTS AI Competition, researchers have been motivated to explore diverse AI approaches in RTS games [29]. In this work, we ran our experiments in a game simulator called FastEcslent, developed for evolutionary computing research in games [30]. Unlike other available RTS-like engines, FastEcslent enables 3D movement, and can run without graphics thus providing simpler integration with evolutionary computing approaches.

We predefined a set of scenarios where each automated player controls a group of units initially spawned in different locations on a map with no obstacles. The entities used in FastEcslent reflect those in StarCraft, more specifically, Vultures and Zealots. A Vulture is a vulnerable unit with low hit-points but high movement speed, a ranged weapon, and considered effective when outmaneuvering slower melee

TABLE I
UNIT PROPERTIES DEFINED IN FASTECSLENT

Property	Vulture	Zealot
Hit-points	80	160
MaxSpeed	64	40
MaxDamage	20	16*2
Weapons Range	256	224
Weapons Cooldown	1.1	1.24

units. A Zealot is a melee unit with short attack range and low movement speed but has high hit-points. Table I shows the details of these properties for both Vultures and Zealots which are used in our experiments. Since our research focuses on micro behaviors in skirmishes, we disabled fog of war and enabled 3D movement by adding maximum (1000) and minimum (0) altitudes, as well as a climb rate constant, r_c , of 2. Comparing to StarCraft, units move in 3D by setting a desired heading (dh), a desired altitude (da), and a desired speed (ds). Every time step δt , a unit tries to achieve the desired speed by changing its current speed (s) according to the acceleration (r_s).

$$s = s \pm r_s \delta t \tag{1}$$

where r_s is the units acceleration, δt is the simulation time step, and $\pm$ depends on whether ds is greater than or less than s. Similarly,

$$h = h \pm r_t \delta t \tag{2}$$

and

$$a = a \pm r_c \delta t \tag{3}$$

Where h is heading, a is altitude, r_t is turn rate, and r_c is climb rate. From speed, heading, and altitude, we compute 3D unit velocity (vel) and position (pos) as follows:

$$\textbf{vel} = (s * cos(h), 0, s * sin(h))$$

$$\textbf{pos} = \textbf{pos} + \textbf{vel} * \delta t$$

$$\textbf{pos}.y = a$$

Here, bold text indicates vector variables, the xz plane is the horizontal plane, the y-coordinate is height, and the unit points along its heading. This type of set-up is important because units micro is governed by physics; that means micro depends on units turning rate, speed and acceleration. The effectiveness of a unit that can turn quickly and attack in all directions is different compared to the effectiveness of a unit that does not have the ability to turn quickly and can not attack in all directions. Furthermore, a unit with high-speed and high acceleration has the ability to flee quickly when outnumbered compared to a unit with less speed and less acceleration. Given a simulation environment within which we can fight battles between unit groups from two different sides, we need an opponent to evolve against. We first describe our representation and then describe how we generate a good opponent to evolve against within this representation. Figure 1 shows the screen shot of our 3D RTS game simulation "FastEcslEnt".

Fig. 1. Screen shot of FastEcslEnt Simulation .

IV. METHODOLOGY

We create several game maps (or scenarios) with two types of units on each side. When we run a fitness evaluation, a decoded chromosome controls our units as they move, using potential fields, towards a target location defined by an influence map. This game-simulation stops when all the units on one side die or time runs out. The simulation tracks the health of units and provides a multi-objective fitness (damage done, damage received) for this chromosome to drive evolution. The rest of this section, describes the scenarios, potential fields, and influence maps used in our work.

Earlier work has shown that evolving (training) on a single map with fixed starting locations for all units did not result in robust micro [10]. We therefore train our units over five different scenarios and measure the robustness of evolved micro on 50 unseen randomly generated scenarios. In this work, randomly generated scenarios means only that units start at different initial positions at the beginning of a fitness evaluation. Scenarios are constructed from "clumps" and "clouds" of entities; defined by a center and a radius. All units in a clump are distributed randomly within a sphere defined by this radius (400 for this paper). Units in a cloud are distributed randomly within 10 units of the sphere boundary defined by the center and radius (also 400).

We created two sides; player1 with 5 Vultures and 5 Zealots and player2 with 5 Vultures and 5 Zealots. The training scenarios are as follows: (a) A clump of player1 versus a clump of player2, (b) A clump of player1 units surrounded by a cloud of player2 units, (c) A clump of player2 units surrounded by a cloud of player1 units, (d) A set of player1 units within range of 250 in all three dimension centered at the origin and a set of player2 units within 250 in all three dimension centered at 650, and (e) the same distributions of units but with the players swapping their centers. Our evaluation function ran each of these five scenarios for every chromosome during fitness evaluation and the value returned by the simulation for each objective is averaged over these scenarios. This results in evolving more reliable micro that can do well under different training scenarios.

Once a scenario starts running, units have to come up with a target location to attack. An influence map determines this

target location.

A. Influence Maps

A typical IM is a grid defining spatial information in a game world, with values assigned to each grid-cell by an IMFunction. These grid-cell value are computed by summing the influence of all units within range, r of the cell. r is measured in number of cells. The IM not only considers units positions in the game world but also includes the hit-points and weapon cool-down of each unit. The influence of a unit at the cell occupied by the unit is computed as the weighted linear sum these factors. A unit's influence thus starts as this weighted linear sum at the unit's cell and decreases with distance from this cell by a factor: I_f. The NSGA-II evolves these parameters and evolving units move towards the lowest IM grid-cell value [2] using potential fields to guide all movement.

B. Potential Fields

We use potential fields to guide unit movement to the target location provided by the IM. Once near the opponent, we would like our units to maneuver well based on the location of enemy units, their health, and the state of their weapons. We define potential field in the form of cd^e where c and e are evolvable parameters and d can be distance, health, or weapons state. We thus define and use attractive and repulsive potential fields for each of these factors. Since the fields for friendly units should be different from the fields for enemy units, we use two such sets of potential fields. Finally, the target location also exerts an attractive potential. Ignoring the target location's potential field, this results in a total of 2 (attractive, repulsive) $\times 3$ (location, health, weapons state) $\times 2$ (friend, enemy) $= 12$ potential fields for guiding one type of unit's movement against an enemy also composed of only one type of unit. We use the same techniques from [2] to convert the vector sum of these potential fields into a desired heading and desired speed and same ranges of value for potential field parameters. Once we move to micro for groups composed from two types of units, the number of potentials fields increases.

We need different potential fields for different types of units because each type of unit treats other types of units differently. For example a friendly melee unit treats enemy melee units differently from enemy ranged units. The friendly melee unit should avoid enemy ranged units and approach enemy melee units. In contrast, a friendly ranged unit can target any enemy unit during skirmishes. Thus we need different potential fields and IM parameters for each type of unit.

Figure 2 shows the four sets of potential fields needed when dealing with groups composed from two types of units where F_1 represents type one friendly units and F_2 represents type two friendly units. For now, we ignore potential fields generated by the target location. Similarly E_1 represents type one enemy units and E_2 represents type two enemy units. A total of 4 (two attractive and two repulsive) fields are required corresponding to F_1, F_2, E_1, and E_2 for each of the following properties: distance, health and weapon state. This results in a total of 4 (attractive, repulsive) $\times 3$ (distance, health, weapons state) $\times 2$ (friend, enemy) $= 24$ potential fields per unit type.

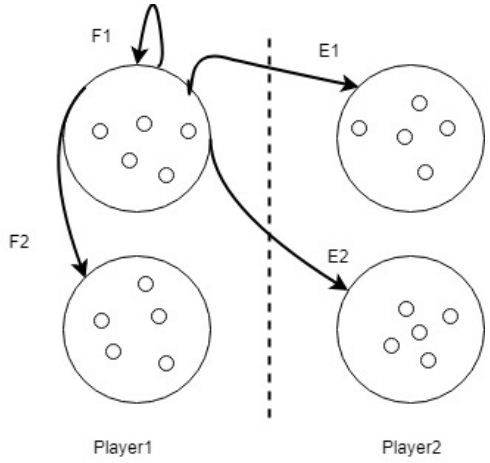


Fig. 2. Potential fields needed for groups composed from two types of units.

The above explanation calculated the number potential fields required for one type of unit. If p represents the number of potential fields, then for one type versus one type $p = 12$. For two types versus two types of units, recall that we needed 24 potential fields of each type of unit. Thus we will need 2 (types of units) $\times 24 = 48$ potential fields required.

Generalizing if we consider skirmishes between n types of units versus n types of units, then the number of potential fields required per type of unit is $n \times p$ and the total number of potential required is $n \times n \times p$. Each potential field has two parameters, thus in total we need $2 \times n \times n \times p$ parameters.

We now consider the potential field exerted by the target location. Each type of unit has its own target location, and hence requires a fixed number of parameter for its target potential field and influence map . Let q be this fixed number of parameters. q grows linearly with n, thus $q \times n$ parameters are required for n different types of units on each side. We can see that Equation 4 gives the total number of parameters required to deal with n different types of units to a side where TNP refers to total number of parameters. Note that the distance potential field parameters are computed more than once, as different types of units are added to each side. We can therefore subtract the i additional potential field parameters from the total number of required parameters.

Thus for two types of units on each side, we need a total of 106 parameters. In real game play, we usually micro with four or five different types of units resulting in 388 or 595 parameters according to Equation 4. This seems feasible to evolve with our micro representation.

$$\text{TNP} = (q + 2pn)n - \sum_{i \in n} 4(i-1) \tag{4}$$

These parameters provide a target location and guide unit movement towards the target. If enemy units come within weapons range of a friendly unit, the friendly unit targets the nearest enemy unit. In our game simulation all entities can

fire in any direction even while moving from one location to other. With a good set of parameters the units evolve effective micro that tries to maximize damage done to enemy units while minimizing damage taken.

Although some work has combined damage done and damage received into one objective to be maximized, we keep the objectives separated and use an evolutionary multi-objective optimization approach to evolve a diverse pareto front. Specifically, we use our implementation of the Fast Non-dominated Sorting Genetic Algorithm (NSGA-II) to evolve a pareto front of micro behaviors for heterogenous groups composed from two types of units. We try to maximize damage done to enemy units while minimizing damage to friendly units. Assume that we normalize damage done and damage received to span the range $[0..1]$, Equation 5 describes our multi-objective optimization problem.

$$\text{Maximize}\left[\sum_{enemies}(D_e), \sum_{friends}(1-D_f)\right] \quad (5)$$

Here, D_e represents damage done to enemy units and D_f represents damage to friendly units. Our objectives are to maximize D_e and minimize D_f respectively. To minimize damage to friendly units we subtract from the maximum damage possible, 1, to also turn the second objective into a maximization objective. This normalized, two-objective fitness function used within our NSGA-II implementation then produces the results described in our results section.

C. Baseline Opponent AI

In order to produce high quality micro behavior, finding a good opponent to play against is crucial. Instead of hand-coding an opponent, we use a two step approach to find a good opponent. First, we generated 30 random chromosomes that we used as opponents and ran NSGA-II against each one of them with population size of 20 for 30 generations. The best opponent is the one that does most damage to friendly units. We thus choose this opponent chromosome that does the most damage as the next opponent. Second, we then run our NSGA-II against this chromosome. The last generation pareto front of NSGA-II provides a diverse set of micro behaviors ranging from fleeing; less damage done and less damage received (0, 1) to kiting; more damage done and more damage received (1, 0). We choose the most balanced performance, closest to (0.5, 0.5), as the next opponent micro and repeat this process five times (five steps). These five steps provide $five$ Balanced Opponent Micro (BOM) chromosomes (BOM1 - BOM5). Since we do not use hand-coded opponent micro, we ran 1000 randomly generated chromosomes against BOM1 through BOM5 to better understand their effectiveness.

Figure 3 shows the performance of 1000 randomly generated chromosomes against BOM1 to BOM5. In the figure, the line marked BOi represents the pareto front of these 1000 random chromosomes against BOMi. That is, the points on the line represent the best performers from among these 1000 random individuals against BOMi The x-axis represents damage done, while the y-axis represents 1 - damage received.

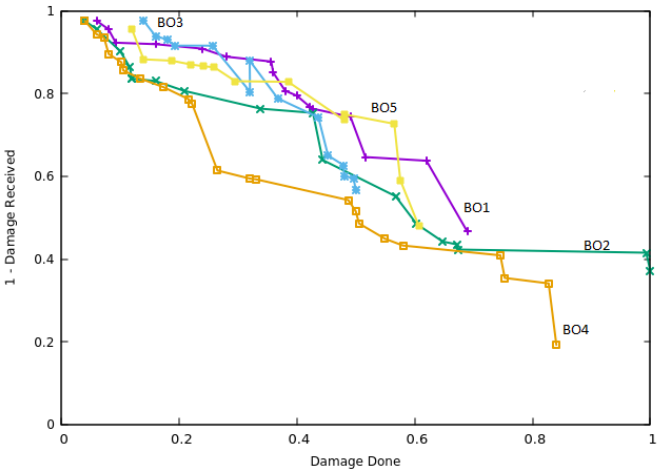


Fig. 3. Pareto front of 1000 random chromosome against BOM1 to BOM5

The point $(1, 1)$ then represents micro that destroys all enemy units and receives no damage. $(1, 0)$ is micro that does destroys all opponents but also loses all friendly units. $(0, 1)$ usually indicates fleeing behavior, friendly units deal no damage and receive no damage. $(0, 0)$ is bad, friendly units did no damage and received maximal damage - micro to be avoided. From the figure, we can see that the line marked as BO4 did worst. This means that the 1000 chromosomes did worst against BOM4, the balanced individual from the last generation pareto front in step four above. Furthermore, this implies that BOM4 is the most difficult balanced opponent to play against and thus, with high probability, a good opponent to evolve against. To confirm that BOM4 would make a good opponent to evolved against, we ran 3750 random chromosomes against BOM4 and against a fleeing individual from the last generation pareto front from step four.[1] Figure 4 shows how these two individuals fare against these new random chromosomes. Clearly these individuals again perform worse against the balanced opponent (BOM4) and we thus choose BOM4 as our final opponent in the experiments described below.

V. Results and Discussion

We use real-coded parameter with simulated binary crossover (SBX) along with polynomial mutation. After experimenting with different values, we settled on the following. Crossover and mutation distribution indexes were both set to 20. We used high probabilities of crossover (0.9) and mutation (0.05) to drive diversity.

A. Pareto Front Evolution of Final Experiment

We evolved micro for groups of 5 vultures and 5 zealots versus an identical opponent also with 5 vultures and 5 zealots. We used a population size of 50 for 75 generations and report results over 10 runs using a different random seed for each run.

[1] 3750 comes from our experiments in the next section where we use a population of 50 running for 75 generations. $3750 = 50 \times 75$.

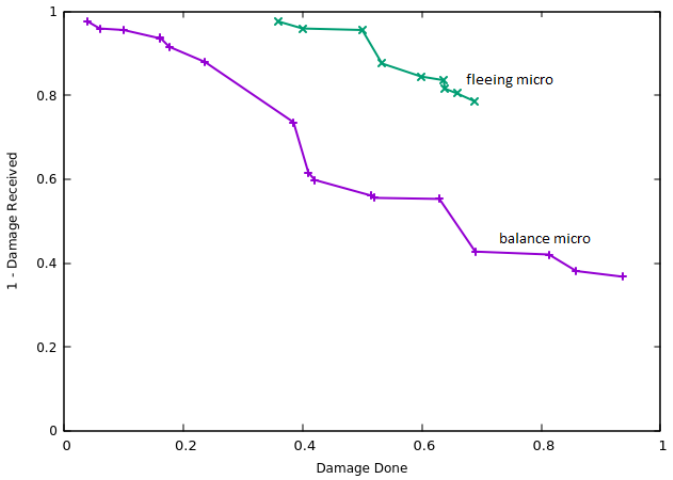


Fig. 4. Comparing the pareto front of 3750 random chromosome against good balanced and good flee-er from step four

Figure 5 shows the evolution of the pareto front at intervals of fifteen (15) generations for one run of our parallelized-evaluation NSGA-II. Broadly speaking, the pareto front moves towards $(1, 1)$ while maintaining representatives along the tradeoff curve for maximizing damage done and minimizing damage received. We can see the maintainence of a diverse set of micro making a diverse set of tradeoffs between damage done and received. These results provide evidence that we can evolve a diverse set of micro tactics that learns to performs well against an existing opponent.

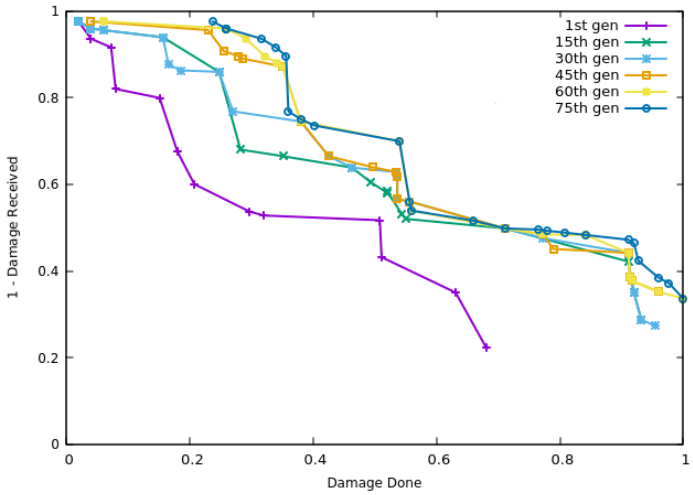


Fig. 5. Micro Evolution for Friendly Unit in Final Experiment

To test the effectiveness of our evolutionary multi-objective optimization approach, we played a balanced individual and a fleeing individual from the 75^{th} generation pareto front against 3750 randomly generated chromosomes. Figure 6 shows how these random chromosomes did against the evolved micro. For comparision, we also ran BOM4 and fleeing micro from last generation pareto front of step four against these random individuals.

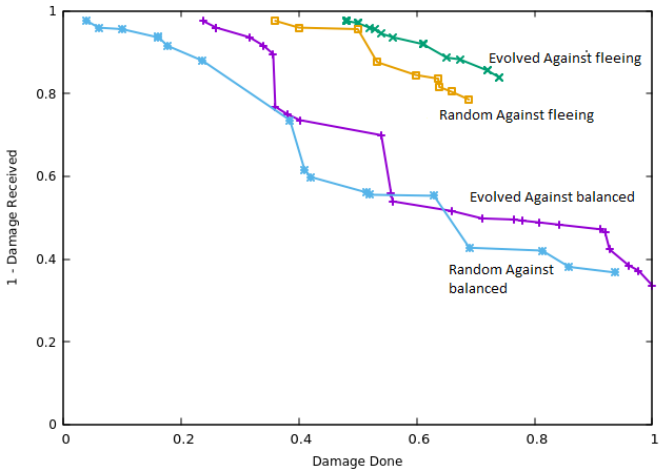


Fig. 6. Comparing evolved micro against 3750 random chromosomes

The figure shows that our evolved balanced individual does better than fourth step balanced individual, and the evolved fleer also does better than the step four fleer. Evolutionary multi-objective optimization's approach to producing a diverse set of solutions along the pareto front leads seems to lead to robust micro. Watching the micro we can see the emergence of kiting, withdrawing, and other kinds of behavior often seen in human game play. Videos of the evolved micro are available at https://www.cse.unr.edu/~rahuld/CIG2018/.

Figure 7 plots the combined pareto front in the first generation over all ten random seeds versus the combined pareto front in the last generation over the ten random seeds. That is, we first did a set union of the pareto fronts in the ten initial randomly generated populations. The points in this union over all ten runs are displayed as purple $+$ for the initial generation (generation 1) points and as green $\times$ for the points in the final generation (generation 75). The figure then shows progress between the first and last generation over all ten runs. We can see that the last generation pareto front produces micro on one extreme on the left (0.02, 0.98) representing a strong tendency to flee, to the other extreme on the right (1,0.25) denoting an aggressive attacking micro behavior. There are a number of solutions near the middle with balanced micro behavior.

To further check the robustness of our evolved micro on the last generation pareto front, we decided to select one balanced, one fleeing, and one attacking example of micro from this last generation and play against BOM4 in a variety of different randomly generated scenarios. In these 50 scenarios, we randomly varied the numbers of zealots and vultures, both between $5 - 10$, and made sure that both sides had identical units. Figure 8 shows results, indicating that the evolved attacking micro (green $\times$s) comes in on the lower right, generally dealing damage while also receiving significant damage. On average over the 50 scenarios, the attacking micro

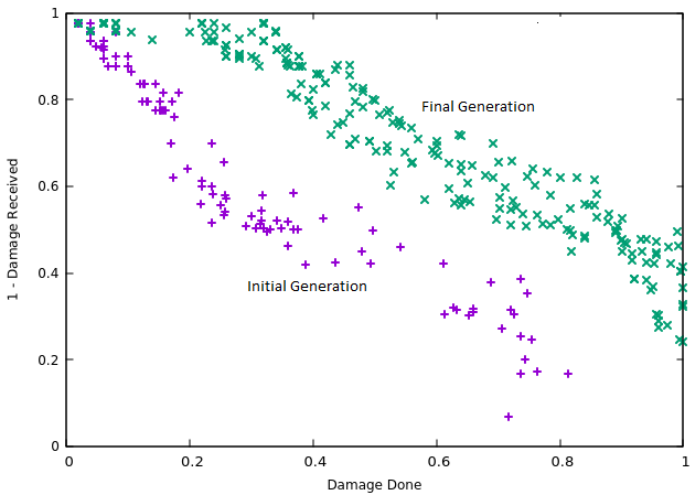


Fig. 7. The initial and final generation pareto front over ten runs for evolved micro

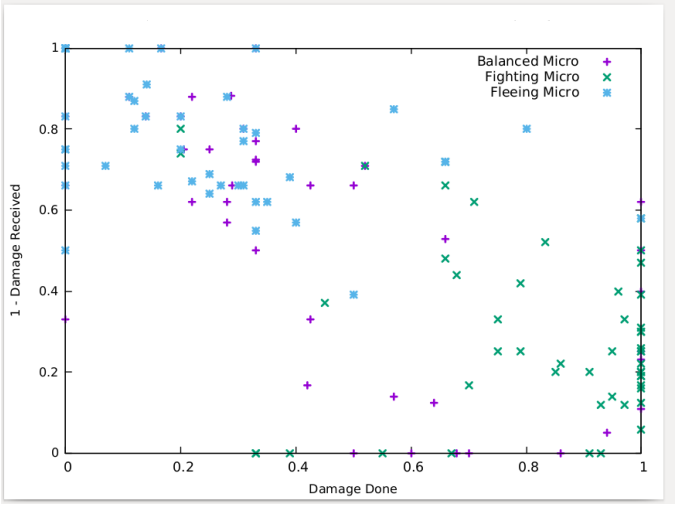


Fig. 8. Robustness of evolved micro on 50 random testing scenarios

leads to objective function values of $(0.812, 0.291)$, while the balanced micro leads to an average of $(0.39, 0.59)$ and the fleeing micro's average fitness comes to $(0.21, 0.79)$.

Finally, we played the evolved attacking micro against larger numbers of opponents. The video at https://www.cse.unr.edu/∼rahuld/CIG2018/Video1.mp4 shows how 5 vultures and 5 zealots controlled by our evolved attacking micro plays against and defeats 5 vultures and 10 zealots controlled by BOM4. The attacking micro manages to destroy all 15 opponent units showing that better micro can win even when outnumbered. A second video at https://www.cse.unr.edu/∼rahuld/CIG2018/Video2.mp4 shows our attack micro controlled 5 vultures and 5 zealots playing against 5 Vultures and 15 Zealots controlled by BOM4. This is an example of the type of effective kiting that evolves over time.

VI. Conclusions and Future Work

This paper focused on extending research in multi-objective optimization and potential fields based representation to evolve micro for groups composed from heterogeneous (two) types of units. We choose a group of ranged and melee units to play against a group of ranged and melee considering damage done and damage received as two objective functions. We use an evolutionary multi-objective optimization approach that maximizes damage done and minimizes damage received to tune influence map and potential field parameter values that lead to winning skirmishes in our scenario.

We can see the emergence of kiting and other complex behavior as the population evolves. The multi-objective problem formulation using the fast non-dominated sorting GA evolves pareto fronts that produced a diverse range of micro behaviors. These solutions not only beat the opponent that they played against to determine fitness, but are robust to different numbers of opponents and can beat an opponent even when outnumbered.

Although this work dealt with two unit types, we would like to extend our work to multiple unit types and to reduce the need for a good opponent to evolve against. As one of the reviewers pointed out, evaluating our evolved micro against a hand-coded opponent should give us a better absolute performance measure. We plan to do so. Since we had to manually co-evolve the opponent in this paper, we also plan to investigate co-evolutionary multi-objective approaches. That is, we would like to use a multi-objective, co-evolutionary algorithm to co-evolve a range of micro that is robust against a range of opposition micro. Finally, we plan to work on the StarCraft -II API to implement our approach and representation to evolve micro to test against human experts.

References

[1] S. Ontaon, G. Synnaeve, A. Uriarte, F. Richoux, D. Churchill, and M. Preuss. A survey of real-time strategy game AI research and competition in StarCraft. *IEEE Transactions on Computational Intelligence and AI in games*, 5(4):1–19, 2013.

[2] S.J. Louis and S. Liu. Multi-objective evolution for 3D RTS micro. *Neural and Evolutionary Computing, arXiv:1803.02943*, 2018.

[3] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA- II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[4] C. Ballinger and S. Louis. Learning robust build-orders from previous opponents with coevolution. *in Proc. IEEE Conf. Comput. Intell. Games*, pages 1–8, 2014.

[5] B. Gmeiner, G. Donnert, and H. Kostler. Optimizing opening strategies in a real-time strategy game by a multi-objective genetic algorithm. *in Research and Development in Intelligent Systems XXIX*, pages 361–374, 2012.

[6] H. Kostler and B. Gmeiner. A multi-objective genetic algorithm for build order optimization in starcraft-II. *KI-Kunstliche Intelligenz*, 27(3):221–233, 2013.

[7] S. Liu, S. J. Louis, and M. Nicolescu. Using CIGAR for finding effective group behaviors in RTS game. *IEEE Conference on Computational Intelligence in Games*, pages 1–8, 2013.

[8] D. Churchill, A. Saffidine, and M. Buro. Fast heuristic search for RTS game combat scenarios. *Interactive Digital Entertainment Conf. in Artificial Intelligence*, pages 112–117, 2012.

[9] S. Liu, S. J. Louis, and M. Nicolescu. Comparing heuristic search methods for finding effective group behaviors in RTS game. *IEEE Congress on Evolutionary Computation*, pages 1371–1378, 2013.

[10] S. Liu, S. Louis, , and C. Ballinger. Evolving effective micro behaviors in real-time strategy games. *IEEE Transactions on Computational Intelligence and AI in Games*, 8(4):351–362, 2016.
[11] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *The international journal of robotics research*, 5(1):90–98, 1986.
[12] S. Liu and S. J. Louis. Comparing two representations for evolving micro in 3d RTS games. *IEEE International Conference on Tools with Artificial Intelligence*, pages 722–789, 2016.
[13] J. Schmitt and H. Kostler. A multi-objective genetic algorithm for simulating optimal fights in starcraft II. *IEEE Conference on Computational Intelligence and Games*, 2016.
[14] C. Miles, J. Quiroz R. Leigh, and S. Louis. Co-evolving influence map tree based strategy game players. *IEEE Symposium on Computational Intelligence and Games*, pages 88–95, 2007.
[15] P. Sweetser and J. Wiles. Combining influence maps and cellular automata for reactive game agents. *Intelligent Data Engineering and Automated Learning-IDEAL*, pages 209–215, 2005.
[16] M. Bergsma and P. Spronck. Adaptive spatial reasoning for turn-based strategy games. *Fourth Artificial Intelligence and Interactive Digital Entertainment Conference*, pages 161–166, 2008.
[17] M. Preuss, N. Beume, H. Danielsiek, T. Hein, B. Naujoks, N. Piatkowski, R. Stuer, A. Thom, , and S. Wessing. Towards intelligent team composition and maneuvering in real-time strategy games. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(2):82–98, 2010.
[18] A. Uriarte and S. Ontan. Kiting in RTS games using influence maps. *Eighth Artificial Intelligence and Interactive Digital Entertainment Conference*, pages 31–36, 2012.
[19] M. Barbuceanu and M. Fox. Cool: A language for describing coordination in multi agent systems. *Proceedings of the First International Conference on Multi-Agent Systems*, pages 17–24, 1995.
[20] N. Jennings. Commitments and conventions: The foundation of coordination in multi-agent systems. *Knowledge Engineering Review*, 8:223–223, 1993.
[21] N. Jennings. Controlling cooperative problem solving in industrial multi-agent systems using joint intentions. *Artificial intelligence*, 75(2):195–240, 1995.
[22] R. Olfati-Saber, J. Fax, and R. Murray. Consensus and cooperation in networked multi-agent systems. *Proceedings of the IEEE*, 65(1):215–233, 2007.
[23] C. Reynolds. Flocks, herds and schools: A distributed behavioral model. *in ACM Digital library Computer Graphics*, 21(4):25–34, 1987.
[24] Y. Chuang, Y. Huang, M. DOrsogna, and A. Bertozzi. Multi-vehicle flocking: scalability of cooperative control algorithms using pairwise potentials. *IEEE International Conference on Robotics and Automation*, pages 2292–2299, 2007.
[25] G. Yannakakis and J. Hallam. Evolving opponents for interesting interactive computer games. *From Animals to Animats*, 8:499–508, 2004.
[26] D. Doherty and C. ORiordan. Evolving tactical behaviours for teams of agents in single player action games. *9th International Conference on Computer Games: AI, Animation, Mobile, Educational and Serious Games*, pages 121–126, 2006.
[27] P. Avery, S. Louis, and B. Avery. Evolving coordinated spatial tactics for autonomous entities using influence maps. *IEEE Computational Intelligence and Games*, pages 341–348, 2009.
[28] P. Avery and S. Louis. Coevolving team tactics for a real-time strategy game. *IEEE Congress on Evolutionary Computation*, pages 1–8, 2010.
[29] Michael Buro and David Churchill. Real-Time Strategy Game Competitions. *Competition Report AI MAGAZINE*, pages 106–108, 2012.
[30] FastEcslent. (2016) Evolutionary computing systems lab, unr. [Online]. Available. http://ecsl.cse.unr.edu/.

Monster Carlo: An MCTS-based Framework for Machine Playtesting Unity Games

Oleksandra Keehl
Design Reasoning Lab
Department of Computational Media
University of California, Santa Cruz
okeehl@ucsc.edu

Adam M. Smith
Design Reasoning Lab
Department of Computational Media
University of California, Santa Cruz
amsmith@ucsc.edu

***Abstract*—We describe a Monte Carlo Tree Search (MCTS) powered tool for assessing the impact of various design choices for in-development games built on the Unity platform. MCTS shows promise for playing many games, but the games must be engineered to offer a compatible interface. To circumvent this obstacle, we developed a support library for augmenting Unity games, and Python templates for running machine playtesting experiments. We also propose ways for designers to use this tool to ask and answer designs questions. To illustrate this, we integrated the library with *It's Alive!*, a game in development by the authors, and *2D Roguelike*, an open source game from the Unity asset store. We demonstrate the tool's ability to answer both game design and player modeling questions; and provide the results of system validation experiments.**

***Index Terms*—machine playtesting, MCTS, restricted play, Unity**

I. Introduction

Human playtesting is an irreplaceable aspect of game development. It can also be logistically cumbersome and creates a significant bottleneck in the design cycle: design, build, test, learn, and redesign. One of the main arguments for machine playtesting is that a simulator can play through games orders of magnitude faster than a human. Thus, it can cover more ground, collect more data, and, in some cases, provide guarantees through exhaustive search.

We present Monster Carlo, a framework for machine playtesting Unity[1] games. Tools based on this framework can gather data on different design variants and playstyles in order to detect imbalance and other effects design changes may have on a player's experience. We set out to apply Jaffe's Restricted Play balance framework [1], to *It's Alive!*, a game in-development by the first author (Figure 2). Contrasting with Jaffe's work, which examined the win–lose outcome of competitive two-player card games, *It's Alive!* emphasizes maximizing the score in a single-player *Tetris*-like game. Exhaustive search is not tractable in the general case of *It's Alive!*, due to the vast number of reachable states. In response, we apply Monte Carlo Tree Search (MCTS) to find input sequences that approximately maximize the player's score.

Monster Carlo aims to bring AI techniques to game developers in the platforms they are already using. In the implementation of our framework, we made an effort to minimize the game code changes required for integration. We also provide result visualization templates in Jupyter Notebook, a tool often used for gameplay data analysis [2].

To test the tool's versatility, we integrated it with a game of a different style and one we did not develop: *2D Roguelike* (Figure 5), the open source game from the Unity asset store.[2]

Monster Carlo is meant to answer a variety of design questions: In *It's Alive!*, how do monster "come to life" conditions affect the achievable high scores? How does a player who collects monsters right away compare to a player who waits until the last moment? In *2D Roguelike*, how does the game dynamic change if we increase both the damage dealt by the zombies and health gained from food pick-ups? How feasible is a no-backtracking player strategy?

MCTS has many variations. We experimented with never re-visiting fully explored branches of the search tree, aiming to increase the number of states explored. We tested different values for the tunable exploration constant in the UCT[3] algorithm, finding the search performs better if the constant is closer to an average score of a human player. We also added an optimization technique of saving the entire playtrace of each playthrough with a new best score [3]. All of these game-agnostic variations are available through parameters of Monster Carlo.

In general, using MCTS for machine playtesting requires the games to be engineered to be compatible with it. The Monster Carlo support library is designed to hook into the Unity engine update cycle and makes this engineering easy. We also created data analysis templates for experiments that take the form of comparing optimized scores between variants.

This paper makes the following contributions:

- A framework for machine playtesting Unity games where instances of the game execute rollouts for MCTS.
- First work to use MCTS at the level of a whole game platform rather than a specific game.
- C# and Python support libraries for adapting a game to support machine playtesting and running experiments.[4]

[1]https://unity3d.com/

[2]https://www.assetstore.unity3d.com/en/#!/content/29825

[3]Upper Confidence Bound-1 applied to Trees

[4]https://github.com/saya1984/Monster-Carlo

978-1-5386-4359-4/18/$31.00 ©2018 IEEE

- Initial experiments that validate the framework and answer design questions about an in-development game.

II. Related Work

In this section, we review the work related to three topics relevant to Monster Carlo: design inquiry with restricted play, MCTS, and frameworks designed to support MCTS.

A. Design Inquiry with Restricted Play

Ludocore is a logical game engine for modeling video games [4]. Smith *et al*. imposed restrictions on the player behavior in order to analyze games created within the Ludocore framework. From Ludocore, we borrow the idea of asking design questions by restricting our player models and observing how these constrained players perform as variations in the design are considered. In Ludocore, games had to be encoded in a specially-designed logic programming language which Ludocore's back-end analysis engine could understand. By contrast, Monster Carlo is meant for integration with Unity games. Such games can dynamically allocate memory, hand-off simulation to a physics library, or perform other computations that would be tedious to model in a purely symbolic framework.

The Restricted Play concept of asking game balance questions by preventing or forcing a player to do certain actions in game was introduced by Jaffe *et al*. [1] and was applied to a two-player, perfect-information game *Monsters Divided*. In their evaluation tool, the authors calculated the optimal strategies for each type of restricted behavior. The game size (five cards per player) allowed for exhaustive search through the entire game tree, foreseeing every possible playthrough. In their Future Work section, the authors state that MCTS is a promising alternative for games whose complexity makes exhaustive search impractical. Because of MCTS's agnosticism to a game's features, it can be used without modification on different restricted players and game design variants. In this paper, we extend Jaffe's restricted play idea, combining it with MCTS and applying it to a new class of games: single player, discrete state games with a larger space of states.

Zook *et al*. [5] follow up on Jaffe's suggestion to use MCTS for analyzing large games. They experimented with *Scrabble* and the *Magic: The Gathering* inspired card game *Cardonomicon*. They used restricted play to simulate player skill levels to see what trends and strategies emerge when players of different skills are pitted against each other. Although *Scrabble* and *Cardonomicon* are naturally imperfect-information games (one cannot be sure which tiles or cards the opponent has until they are played), Zook *et al*. work with determinized, perfect-information variations of these games (explained further in section II-B). *It's Alive!* is also a nondeterministic game (the player does not know which piece will be randomly dropped next), and we use the determinization strategy by fixing the game's random seed value.

Holmgrd *et al*. [6] used genetic algorithms to evolve custom evaluation functions for use with MCTS to simulate different playstyles in the game MiniDungeons 2. They used the evolved personas to assess feasibility of each playstyle in different level layouts. This approach can be combined with Monster Carlo, as it allows the user to implement custom evaluation functions for use within simulations and final score computation.

B. MCTS

Monte Carlo Tree Search (MCTS) is a heuristic search algorithm which selectively explores the tree of possible moves [3]. It assesses the potential of each move by averaging the scores of simulated random playouts from the current point in the game until a terminal state. Although MCTS denotes a broad family of algorithms, the most common, UCT, has a single tunable parameter: the balance of advancing the more promising branches of the tree with exploring the paths less traveled. MCTS is a relatively simple algorithm and can be used with a multitude of decision problems without the need for game-specific heuristics. It has been successfully applied to *Go* [7], *Tetris* [8], *Scrabble* [5] and other games. Most recently, it was used in the creation of AlphaZero [9], the latest world champion in *Go*. In the rest of this paper, we use the general term MCTS to refer to the specific instance of UCT in the Monster Carlo framework.

MCTS relies on building a tree of the possible moves in each state. In games with a random element, such as *Tetris* or *Scrabble*, the search was performed using a predetermined sequence of pieces. In the determinized version of the game, the automated player's goal is simply to find the single best sequence of moves that maximizes the final score. Without determinization, the much more difficult goal is to devise a policy that maximizes the expected score averaged over all possible random elements in the game. We used determinization, as it is sufficient to answer the design questions Monster Carlo is intended to answer. One of the main differences in our application of MCTS is that the games described above all had a win/lose condition (even *Tetris*, as it was played competitively), while the games described in this paper focus on the highest score attained.

C. Environments that support MCTS

The Video Game Definition Language (VGDL) [10] is a representational language for modeling videogame mechanics and level designs. The General Videogame Artificial Intelligence (GVG-AI)[5] project provides an interpreter for VGDL games which exposes an MCTS-compatible forward model. Although VGDL has been used to model games inspired by many different kinds of pre-existing videogames, it cannot integrate with the original implementations of any of these games. Like Ludocore, GVG-AI tools can only understand games expressed in a specialized language. By contrast, Unity games can make unrestricted use of the general purpose C# programming language used for Monster Carlo.

OpenAI Gym[6] is a testbed for AI. It includes environments which provide state information, pixel data and rewards in response to an agent's action. It can integrate with commercial

[5]http://www.gvgai.net

[6]https://gym.openai.com/

ROM implementations of many Atari games and can have algorithms learn to play directly from pixel or memory data, rather than the simplified game state abstraction used in VGDL. A growing number of environments are available for AI experimentation. A related effort, OpenAI Universe,[7] aims to allow integration with an even wider array of gameplay-like activities, even including a mock travel arrangement task based on interaction with complex websites. Although these frameworks allow MCTS-style algorithms to play a very wide variety of games, they force interaction with the game at the lowest level of interaction common to all of them: reading pixels or memory bytes and injecting keyboard and mouse actions. By contrast, Monster Carlo is intended to give designers control of the level of abstraction used by MCTS including high-level game actions (e.g. directly playing a card rather than clicking somewhere to select a card). This is important for making Monster Carlo's analysis useful for game developers rather than AI researchers.

All of these systems deal with the representations of game states and actions differently. VGDL uses a data structure for tracking the abstract state of the game and a list of interactions between game objects as action representation. OpenAI Gym uses screenshot pixel data and memory contents for state representation and low-level keyboard and mouse events as actions. Monster Carlo does not represent a game state beyond the sequence of moves needed to reach it, and a way of asking the framework to make discrete micro-decisions which assemble into high-level actions (further described in III).

D. Unity and Unity Machine Learning Agents (ML-Agents)

Unity is a powerful game engine available for free for private use, which makes it popular with independent developers. Unity does not directly support integration with MCTS. To change this, it is necessary for the game to communicate what actions are possible at any moment and provide a way for some AI system to select and apply one of those actions. Additionally, there needs to be a way of communicating a score to be optimized to the AI system. As the level of granularity used to model player choices and the notion of score to be optimized are specific to the game being designed (and even specific to certain design questions being asked of that game), these cannot be provided directly at the level of the Unity platform. In response, Monster Carlo aims to offer the designer a minimal-effort way of expressing game-specific concerns on top of the Unity platform.

Unity ML-Agents[8] is a plugin meant to enable using games and simulations to train agents via various machine learning methods. While both Unity ML-Agents and Monster Carlo involve running many thousands of simulated play traces, Monster Carlo focuses on summarizing those trace for immediate review by designers rather than producing a trained behavioral policy as a side effect. As a result, Monster Carlo has very few parameters to adjust and does not require defining a neural network architecture or other policy representation.

[7]https://blog.openai.com/universe

[8]https://github.com/Unity-Technologies/ml-agents

III. System Design

The Monster Carlo framework consists of four major parts (Figure 1). The integration modifications to the game and specifications for the design experiment (the green parts on Figure 1) have to be written by the game designer, while everything else is provided through the Monster Carlo tool.

We used MCTS as the main search algorithm for our experiments, but the algorithm can be changed without adjusting the game or the design experiment notebooks.

A. Experiment setup and result visualization

The user-facing element of the Monster Carlo framework is a sample Jupyter Notebook for running the experiments and visualizing the results. This includes the game process factory. The user must provide a function that can be called to start an instance of the game, and the game must be compiled with the C# support module (see III-C). The customized process factory can be used to pass experiment-specific configuration data to the game. For example, one can configure it to start in a certain mode optimized for analysis, or arrange for the execution of the game to happen on a remote cluster of machines. The experiment results are returned as an object, which can be saved in a file at the end of each experiment and later used for analysis and visualization. We used matplotlib[9] to visualize the results. All the experiment result graphics in this paper were obtained through this method.

B. Python support module

This module contains the implementation of the MCTS algorithm in Python. Upon the termination of the experiment, it returns an object which contains the search tree and any additional data. This output object can be trivially modified to keep track of additional metrics. In our experiments, we kept track of the growth of the highest seen score over the rollouts, but we could, for example, have as easily kept track of the number of monsters collected during a playthrough. In a narrative oriented game, we might track the fraction of dialog content seen or tally which endings where reached.

The tool supports running multiple instances of the game to significantly speed up the search (see Section V-A for the results). It takes as an argument the number of rollouts and additional optional arguments that include the UCT constant value, the number of parallel workers, terminal branch treatment, saving of the best path option, and a callback function, which can be used to implement custom logging. These are passed to the game instances via environment variables.

C. C# support module

The C# module must be added to the game project. The module takes in the environment arguments at the start of the experiment and communicates with the Python module through a TCP socket. It receives the most promising path prefix determined by the MCTS algorithm from the Python module at the beginning of each playthrough. Each time a

[9]https://matplotlib.org

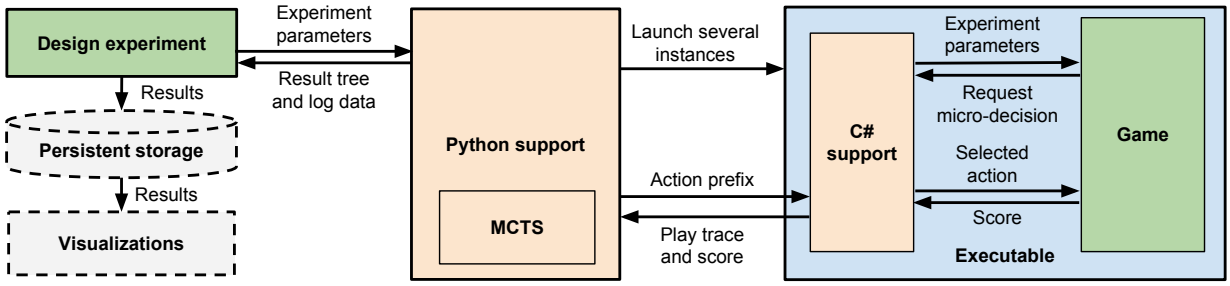


Fig. 1. High level architecture of the Monster Carlo framework. The specification of the design experiment (in Python) and the game code itself (in C#) are project-specific, while the other elements are provided by the framework.

decision must be made in game, the game tells the module how many legal moves are available, and the module makes a choice without needing to know what those moves are. If there are pre-determined moves in the path prefix, the module feeds those back to the game one at a time. When the end of the path is reached, the module continues by making random choices. A custom heuristic can be optionally be expressed in C# by providing an array of action selection weights to be consulted during the rollout phase. When the play session is over, it reports the final score to this module, which sends the full action path, the final score, and any other information the designer specified, back to the Python module.

D. Modifications to the game

The designer must implement micro-decisions and scoring: the game must determine legal moves at each step, request an index of a move to take, and apply that action. When the game reaches a terminal state, it must provide the score to the support module. If random elements are present, each playthrough needs to use the same random seed. We also recommend creating a headless, no-graphics mode for the game, as it can significantly speed up the playthroughs on some platforms.

The game also needs an experiment mode to be able to replace the user's input with decision requests to the C# module. Launching the game in the experiment mode can include skipping menu screens and disabling smooth movements. To optimize the running time, we recommend adding an ability to reset the game after the terminal state is reached, so that the application doesn't have to be re-launched for each fresh playthrough.

Additionally, if the designer wishes to conduct Jaffe-style restricted play experiments, they will have to implement the player models (which may limit available actions before the C# module is queried for a choice). They can also implement a way to switch between the game design variations. The space of design variants considered can be as flexible as the user wants, as long as they can specify those variants in Python and communicate them to the game's executable.

IV. Experiments

This section describes example experiments we conducted using the Monster Carlo framework. They show how to use the restricted play methodology to ask design questions for two games: *It's Alive!* and *2D Roguelike*.

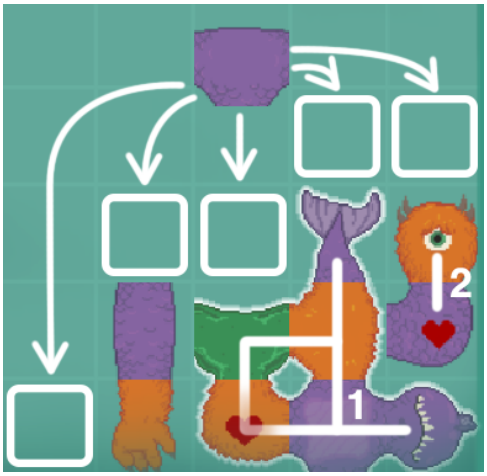


Fig. 2. Possible actions in this state include landing the falling piece in one of the five columns in one of four orientations, or collecting one of the two living monsters.

A. It's Alive!

It's Alive! (see Figure 2) is a *Tetris*-style game where the player controls the position and orientation of pieces falling from the top of the board. Player loses if the pieces pile up to the very top of the board. Rather than trying to make simple horizontal lines of pieces as in *Tetris*, the player must form arrangements of pieces that represent monsters. A monster comes to life when it minimally contains a head piece and a heart piece. At this point, the player may choose to collect it to free up space, or continue building it up. Bonus points are awarded based on the size and color coordination of each monster. If there are several moving monsters on screen, the player can choose which one to collect by shifting the highlight from one monster to another. The player aims for the highest score by animating and collecting five monsters.

The player actions consist of rotating the falling block, moving it left or right, or quick-landing it. The player can also cycle the highlighter through living monsters or delete the currently highlighted monster. Thus, at any point, she has four to six possible keyboard-level actions: rotate, move

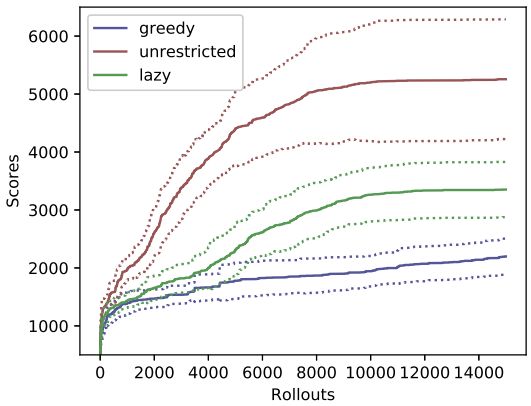


Fig. 3. Highest score achieved for *greedy*, *unrestricted* and *lazy* player models.

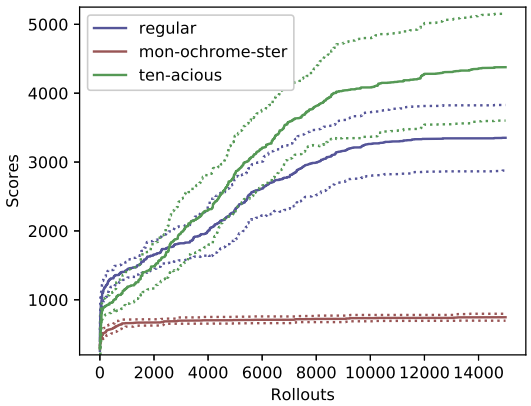


Fig. 4. Highest score for *regular*, *mon-ochrome-ster* and *ten-acious* designs.

left, move right, quick-land, cycle highlighter, collect monster. Some of those actions could be repeated indefinitely without affecting the game state, meaninglessly expanding the scope of the search for Monster Carlo to perform. To avoid this, we use micro-decisions to model only those choices that resulted in meaningful state change. The OpenAI Gym would have forced a mode of interaction at the level of keyboard inputs, whereas Monster Carlo allows the flexibility to focus the analysis on the level of details the designer cares about. Instead of ability to move the block left or right any number of times, the artificial player simply chooses whether to collect one of the living monsters or to land the piece in any orientation in an open column. Similarly, cycling the highlighter is not considered an action, instead, collecting any of the living monsters in the current state is considered a legal action, regardless of the highlighter position (Figure 2). With this new definition of action in mind, the player has 20 or more possible actions at every moment. That is five possible columns times four landing orientations, plus one collect action per living monster. On a 5x7 playfield, this makes exhaustive search computationally intractable due to the vast number of possible combinations.

1) Playstyle experiments: Like *Tetris*, *It's Alive!* has many quick game-over states resulting from piling pieces in the same column and reaching the ceiling while most of the playfield is still empty. To prevent Monster Carlo from wasting time exploring these dead-end scenarios, we prevented all player models from placing a piece that would end the game if a non-game-ending move was possible, such as placing a piece somewhere else or collecting a monster. We did this by excluding the game-ending moves from the list of available actions within the game. No changes to the Monster Carlo framework were required to express this more focused analysis.

We used factored actions for most of *It's Alive!* experiments. Each turn, the player makes a sequence of micro-decisions. First: Should I land the current piece or collect a monster? Next, if I chose to land a piece: Which column should I land it in? Finally, in which orientation should I land the piece?

We experimented with three player styles. The *greedy* player collects the monsters as soon as they came alive. The *lazy* player collects a monster only if the game would otherwise end. The *unrestricted* player is free to collect at any point.

Figure 3 shows that the *lazy* player did best, while the *greedy* player performed the worst. The p-values designating statistical significance of the difference between the scores of each pair of the results ranges from 3.4e-08 to 4.5e-07.[10]

These results show that deciding when to collect a monster is a meaningful choice for the player. Notably, while technically nothing prevented the *unrestricted* player from achieving the same results as the *lazy* player, presenting it with an opportunity to collect the living monster at every step slows down the search progress. This is a reminder that all results from MCTS are approximations computed within a fixed computational budget, so they cannot be trusted with the same level of certainty as in the exhaustive search results in Jaffe's original Restricted Play work. Nevertheless, large score gaps can provide a signal that a designer should look deeper into the specific playtraces found by MCTS that illustrate specific styles of play in action. For this reason, it is important that Monster Carlo returns the resulting tree, not just the aggregate statistics. The user may decide to replay the highest scoring play trace in a mode with more detailed analytics turned on to gain deeper insight into the impact of playstyle difference that the tool discovered.

2) Design variants: We considered three game design variants. The *regular* design follows the rules outlined above. The *mon-ochrome-ster* design considers two pieces within a monster connected only if they are of the same color. In the third variant, *ten-acious*, the monster only comes to life if it consists of at least ten pieces.

The results (Figure 4) show that the *mon-ochrome-ster* mode is much harder than the other two, and affords for a lower maximum score. Counter-intuitively, the *ten-acious* design variant,

[10]Here and in all other experiments, statistical significance is judged according to the single-sided Mann-Whitney U test applied to the highest score achieved within the rollout limit. Each experiment involves 20 independent replicates of each condition.

Fig. 5. Screenshot of Unity tutorial game *2D Roguelike*.

which places a restriction on the player and thus, makes for a harder game, led to higher scores than those Monster Carlo achieved in the *regular* design. Both experiments were run with the same random seed, so nothing prevented the *regular* design player from building monsters of ten blocks or more. The progression of the highest score seen across the rollouts in Figure 4, shows that the *regular* design scores are higher initially, but are quickly overtaken by those seen in *ten-acious*. We believe this is caused by the restrictions in *ten-acious*, which prevented the search exploring the frequent collection of smaller monsters. As before, Monster Carlo does not replace the user's judgment of game design alternatives, but it can gather specific evidence that helps the user make that judgment for themselves.

B. 2D Roguelike

Note that because we are the developers of both Monster Carlo and *It's Alive!*, it is possible that we have over-specialized the framework for analysis of games very much like *It's Alive!*. In this section, we consider the integration effort and results from experiments with a game that we did not make ourselves, nor considered during the primary development of the Monster Carlo framework.

2D Roguelike (Figure 5) is an open source official tutorial game for the Unity game engine.[11] It is grid and turn based: zombies get to take a step for every two steps the player takes. The player starts at the lower left corner of the field and the goal is to reach the exit in the upper right corner, signifying he has survived another day. The game is over when the player runs out of food points and the final score is the number of days the player has survived. One food point is lost for every move and several are lost in case of zombie attacks. The food points can be replenished by picking up food items. The levels are laid out randomly. The number of zombies is a function of the number of days survived, gradually increasing. At any point, the player may choose to go up, down, left or right. Each of these actions results in a state change, as the food points go down even if the player attempts to walk through a wall and does not actually move.

[11]https://www.assetstore.unity3d.com/en/#!/content/29825

1) Playstyle experiments: For *2D Roguelike* we factored the actions into a choice of moving toward or away from the exit, and then deciding whether the move is lateral or vertical. For the first player, as a simple heuristic, we used Monster Carlo's capability for weighted choice to make the player more likely to move toward the exit in the rollout phase of MCTS. The second player was restricted to only move toward the goal. After 30,000 rollouts, the *forward-only* player achieved statistically significantly higher scores (p = 3.3e-08).

Due to the game mechanics, while the *forward-only* player has a short-term advantage of a powerful heuristic, it would eventually come to a hard limit, as it is impossible to pass some levels without backtracking to avoid the zombies. In this game, while the player can break through inner walls, it is impossible to kill the zombies. If the player runs into one and cannot back away, it will eventually kill him. Given enough time, we believe the *unrestricted* player would outperform the *forward-only* player. However, this would take too long to be practically feasible for playtesting. Another option would be to increase the bias with which the *unrestricted* player would select the forward motion vs. backtracking. This would help get more realistic scores faster without imposing the forward-only restriction.

2) Design variants: We compared the game's default configuration with one where both the damage dealt by the zombies and food gained from pick-ups were increased by 50 percent. The results from this *high stakes* design variant were statistically significantly higher (p = 4.8e-07). From this, one could conclude that the *high stakes* variant of the game is easier to play for any given score threshold.

V. Framework Validation

The original design of *It's Alive!* has a 5x7 playfield, which makes for a large search space with the average branching factor of 20 and depth of at least 36 (if no monsters are collected). This leads to longer rollouts and slower depth-wise exploration rate. For the framework validation experiments we reasoned that having a smaller playfield would allow us to run experiments faster while still demonstrating relative differences between performance of Monster Carlo with different parameters. We built a smaller *It's Alive!* with a 3x5 playfield and only three monsters required for the win. A typical human player score for this game is 1200-1400 points. Unless stated otherwise, the experiments were run with an unrestricted player, 24 parallel workers, factored actions, cut-off terminal branch setting, and the exploration parameter in the UCT algorithm set to 1000.

A. Parallel vs. Single Thread

The classic MCTS updates the tree after each rollout and uses the updated tree to chose the next move. With instances of the game running in parallel, the tree is updated each time a playthrough is completed, and the next move is selected with

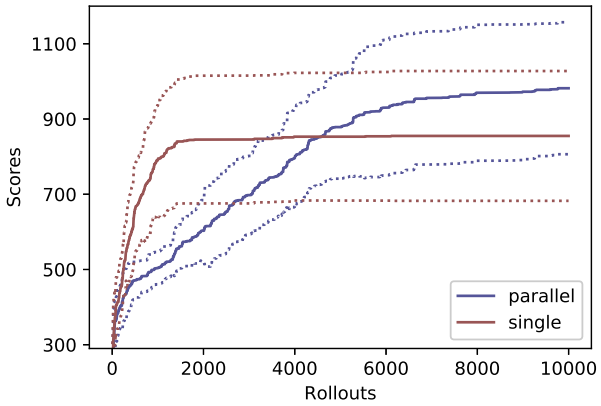


Fig. 6. Highest score achieved for a single worker and 24 workers in parallel on a 3x5 board (20 replicates). The rollouts completed approximately 20 times faster in the parallel case, better results in less wall-clock time.

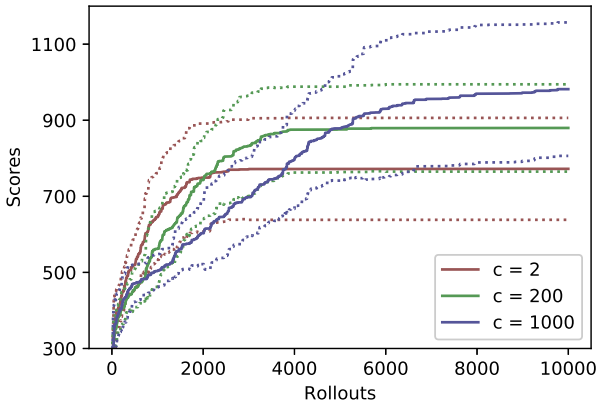


Fig. 7. Highest score achieved with different values of the UCT exploration constant (20 replicates)

the results from several other in-flight rollouts still unknown. This is similar to the tree parallelization with global mutex approach described by Chaslot *et al*. [11]. We wanted to see if there was a large drop in the tool's effectiveness at the cost of the speed. We ran two experiments with 30,000 rollouts. The first set used 24 parallel workers, and the second used a single worker. The parallel experiment achieved higher scores (Figure 6) with statistical significance (p = 0.02). However, the bigger difference is in the duration. The single-thread experiment lasted eight hours and 20 minutes while the parallel experiment took about 24 minutes. Initially, the parallel experiment took more rollouts to get to the same scores as the single-thread experiment. This leads us to extrapolate that parallel workers have a diversifying effect on MCTS. While this initially leads to lower scores, it also makes it less likely for the search to get bogged down in unproductive territory.

B. Terminal Branch Treatment

We noticed that MCTS tended to explore the same branch and get stuck in local maxima, though much better paths were available. We tried two ways around it. One was to increase the UCT constant, traditionally set to 2. The other way was to prevent the tree from revisiting branches marked as terminal.

We ran two experiments, with 30,000 rollouts each, one with no special treatment of terminal nodes and branches, and the other that would mark fully explored sections of the tree as terminal and ignore them during the optimal path selection. The results for these experiments showed no statistically significant difference between the highest scores achieved or the number of nodes explored. We hypothesize that this is largely due to the fact that the depth of our test game was too great, and so the terminal branch treatment did not come into play to a significant degree.

Notably, not revisiting terminal branches allows for exhaustive search on smaller fields. Earlier in this project, we ran tests on *It's Alive!* with a 2x3 grid. We expected the maximum score to be 250 points but found that one of the branches achieved 290 points. This led to the discovery of a bug that only manifested if the monster pieces were positioned in one specific way. After fixing the bug, we were able use Monster Carlo to exhaustively verify the fix by running the same test with cut-off terminal branches and observe that no branches scored higher than 250 points.

C. UCT Constant

The UCT exploration constant (c) regulates how much MCTS focuses on exploring the most rewarding paths vs. exploring new areas. Because MCTS is usually applied to games with a win/lose outcome and the reward values ranging from 0 to 1, we hypothesized that when applied to a game where the reward value is the range of possible high scores, the UCT constant should be closer to a score you would expect from a moderately proficient player. We obtained this score by manually playing the game with the same random seed.

We ran three experiments with respective UCT constant values set to 2, 200 and 1000. The results in Figure 7 demonstrate that Monster Carlo did best with c = 1000, which was closer to the expected score of 1300. The results were statistically significant for comparison of c = 2 and c = 1000 (p = 0.0002), and c = 200 and c = 1000 (p = 0.01). While the experiment with a lower c got slowed in local maxima fairly early on, the scores corresponding to the higher c continued growing due to the search's higher emphasis on exploration.

D. Experiment Speedup Techniques

If the MCTS rollouts happened at the game's normal play speed, each of the aforementioned experiments would take days to complete. Therefore, we employed a number of speedup techniques. Game-side changes included setting Unity's framerate to maximum value, replacing smooth movements with instant jumps and disabling all artificial delays (such as waiting half a second between accepting player inputs). This increased the experiment run speed by a factor of ten. Game-agnostic changes consisted of running parallel search with 24 workers (speedup by a factor of 20) and running

on a server-class machine without graphics (speedup by a factor of eight). The combination of all these allowed us to run experiments at approximately 1600 times faster than the original. Distributing the rollouts across several server-class machines would allow even greater speedups.

VI. Conclusion

We presented Monster Carlo, an MCTS-based tool that can be integrated with the Unity game engine and be used to perform machine playtesting of in-development games; conducted a number of framework validation experiments, which showed merit in adjusting the UCT constant, using parallel processing when performing rollouts, and applying special treatment to terminal nodes and branches. Monster Carlo was integrated with two games: our in-development game *It's Alive!* and an official Unity tutorial game, *2D Roguelike*. The integration with *2D Roguelike* required fewer than 100 lines of code. We also presented results of several experiments run on both games, exploring restricted player models and design variations.

Obtaining reasonable results from MCTS on a complex game takes time, but so does making meaningful changes. Some modifications, like restricted player models or limited variety of pieces, can be added to a game fairly quickly. Larger changes, like introducing a new type of block to the game or adding heuristics to a player model, usually take much longer. With this in mind, even if a set of experimental replicates takes over an hour to run, it can be considered an acceptable turn-around time, as the results will likely be in before the next model is ready for testing. Additionally, the independent runs of MCTS are extremely parallelization-friendly.

Having a reference score helps with setting an appropriate UCT constant value to guide MCTS toward better results. A reference score can be provided by the game designer, or someone familiar with the game, who can play one or two games to provide a baseline score. This score can be helpful for setting the UCT constant, as well as interpreting the MCTS results: if its best scores are much lower than what a casual player can get, it indicates that MCTS needs tuning.

A severely limited player model can still provide information. In early stages of this project, we ran experiments on an even larger *It's Alive!* playfield of 6x8. We used an unrestricted player model, and one that did not rotate the pieces. We experimented with lowering the number of different monster colors. The size of the field led to a very wide tree that never had a chance to explore very deeply and resulted in chaotic and mostly very low scores for the unrestricted player. However, the non-rotating player, whose actions were limited by a factor of four on every step, was capable of reaching more stable scores in the same number of rollouts. The scores made it evident that the no-rotation player got significantly higher scores when fewer monster colors were present (from an average of 1600 to an average of 2100). This is an obvious example, since having fewer colors means it is more likely to get two blocks of the same color next to each other. However, it showed that even a severely limited player model is capable of providing information about design variants.

Much work remains to be done around Monster Carlo, as the scores it achieves in a reasonable time still fall short of human results. In the current setup, the search algorithm has no representation of game state beyond the action sequence, so it cannot transfer experience gained down one sequence of moves to another if they differ by even a single move. Reinforcement learning algorithms such as those used in AlphaZero can distill knowledge gained during MCTS rollouts into value-estimation and rollout-policy networks that can be applied to states that have never been explored before. We believe that borrowing some ideas from frameworks like OpenAI Gym (such as representing game state with universal data structures like screenshot pixel arrays or memory byte arrays) could help a generic search algorithm learn a much better default action policy than even the human user could provide. However, even with its current shortcomings, Monster Carlo is capable of providing usable feedback. With the convenient experimental setup in Jupyter Notebook, our hope is that new kinds of machine playtesting experiments can by invented and executed easily. Nelson *et al*. [12] and Isaksen [13] suggest several strategies for understanding game artifacts, some of which could be implemented using Monster Carlo.

References

[1] A. Jaffe, A. Miller, E. Andersen, Y.-E. Liu, A. Karlin, and Z. Popović, "Evaluating competitive game balance with restricted play," in *Proc. of the Eighth AAAI Conf. on Artificial Intelligence and Interactive Digital Entertainment*, ser. AIIDE'12, 2012, pp. 26–31.

[2] S. Martinelli, "Starcraft ll replay analysis with jupyter notebooks." [Online]. Available: https://github.com/IBM/starcraft2-replay-analysis

[3] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of monte carlo tree search methods," *IEEE Trans. on Comp. Intel. and AI in Games*, vol. 4, no. 1, pp. 1–43, March 2012.

[4] A. M. Smith, M. J. Nelson, and M. Mateas, "Ludocore: A logical game engine for modeling videogames," in *Proc. of the 2010 IEEE Conf. on Computational Intelligence and Games*, Aug 2010, pp. 91–98.

[5] A. Zook, B. Harrison, and M. O. Riedl, "Monte-carlo tree search for simulation-based strategy analysis," in *Proceedings of the 10th Conference on the Foundations of Digital Games*, 2015.

[6] C. Holmgrd, M. Green, A. Liapis, and J. Togelius, "Automated playtesting with procedural personas with evolved heuristics," pp. 1–1, 02 2018.

[7] S. Gelly and D. Silver, "Achieving master level play in 9×9 computer go," in *Proc. of the 23rd AAAI Conf. on Artificial Intelligence*, 2008.

[8] C. Zhongjie, "Playing tetris using bandit-based monte-carlo planning," in *AISB 2011: AI and Games*, 2011.

[9] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of go without human knowledge," *Nature*, vol. 550, pp. 354–359, 10 2017.

[10] T. Schaul, "A video game description language for model-based or interactive learning," in *2013 IEEE Conference on Computational Inteligence in Games (CIG)*, Aug 2013, pp. 1–8.

[11] G. M. B. Chaslot, M. H. Winands, and H. J. van Den Herik, "Parallel monte-carlo tree search," in *Proc. of International Conference on Computers and Games*, September 2008, pp. 60–71.

[12] M. Nelson, "Game metrics without players: Strategies for understanding game artifacts," 2011. [Online]. Available: https://aaai.org/ocs/index.php/AIIDE/AIIDE11WS/paper/view/4114

[13] A. Isaksen, D. Wallace, A. Finkelstein, and A. Nealen, "Simulating strategy and dexterity for puzzle games," in *IEEE Conference on Computational Intelligence and Games*, Aug. 2017.

Deep RTS: A Game Environment for Deep Reinforcement Learning in Real-Time Strategy Games

Per-Arne Andersen
Department of ICT
University of Agder
Grimstad, Norway
per.andersen@uia.no

Morten Goodwin
Department of ICT
University of Agder
Grimstad, Norway
morten.goodwin@uia.no

Ole-Christoffer Granmo
Department of ICT
University of Agder
Grimstad, Norway
ole.granmo@uia.no

***Abstract*—Reinforcement learning (RL) is an area of research that has blossomed tremendously in recent years and has shown remarkable potential for artificial intelligence based opponents in computer games. This success is primarily due to the vast capabilities of convolutional neural networks, that can extract useful features from noisy and complex data. Games are excellent tools to test and push the boundaries of novel RL algorithms because they give valuable insight into how well an algorithm can perform in isolated environments without the real-life consequences. Real-time strategy games (RTS) is a genre that has tremendous complexity and challenges the player in short and long-term planning. There is much research that focuses on applied RL in RTS games, and novel advances are therefore anticipated in the not too distant future. However, there are to date few environments for testing RTS AIs. Environments in the literature are often either overly simplistic, such as microRTS, or complex and without the possibility for accelerated learning on consumer hardware like StarCraft II. This paper introduces the Deep RTS game environment for testing cutting-edge artificial intelligence algorithms for RTS games. Deep RTS is a high-performance RTS game made specifically for artificial intelligence research. It supports accelerated learning, meaning that it can learn at a magnitude of 50 000 times faster compared to existing RTS games. Deep RTS has a flexible configuration, enabling research in several different RTS scenarios, including partially observable state-spaces and map complexity. We show that Deep RTS lives up to our promises by comparing its performance with microRTS, ELF, and StarCraft II on high-end consumer hardware. Using Deep RTS, we show that a Deep Q-Network agent beats random-play agents over 70% of the time. Deep RTS is publicly available at https://github.com/cair/DeepRTS.**

***Index Terms*—real-time strategy game, deep reinforcement learning, deep q-learning**

I. Introduction

Despite many advances in Artificial Intelligence (AI) for games, no universal Reinforcement learning (RL) algorithm can be applied to complex game environments without extensive data manipulation or customization. This includes traditional Real-time strategy games (RTS) such as WarCraft III, StarCraft II, and Age of Empires. RL has recently been applied to simpler game environments such as those found in the Arcade Learning Environment [1](ALE) and board games [2] but has not successfully been applied to more advanced games. Further, existing game environments that target AI research are either overly simplistic such as ALE or complex such as StarCraft II.

RL has in recent years had tremendous progress in learning how to control agents from high-dimensional sensory inputs like images. In simple environments, this has been proven to work well [3], but are still an issue for complex environments with large state and action spaces [4]. The distinction between simple and complex tasks in RL often lies in how easy it is to design a reward model that encourages the algorithm to improve its policy without ending in local optima [5]. For simple tasks, the reward function can be described by only a few parameters, while in more demanding tasks, the algorithm struggles to determine what the reward signal is trying to accomplish [6]. For this reason, the reward function is in literature often a constant or single-valued variable for most time-steps, where only the final time-step determines a negative or positive reward [7]–[9]. In this paper we introduce Deep RTS, a new game environment targeted deep reinforcement learning (DRL) research. Deep RTS is an RTS simulator inspired by the famous StarCraft II video game by Blizzard Entertainment.

This paper is structured as follows. First, Section II and Section III thoroughly outlines previous work and central achievements using game environments for RL research. Next, Section IV introduces the Deep RTS game environment. Section V presents the Deep RTS performance, a comparison between well-established game environments and Deep RTS, and experimental results using Deep Q-Network as an agent in Deep RTS. Subsequently, Section VI concludes the contribution of this paper and outlines a roadmap for future work.

II. Related Game Environments

There exist several exciting game environments in the literature that focus on state-of-the-art research in AI algorithms. Few game environments target the RTS-genre. One the reason may be because these environments are by nature challenging to solve, and there are few ways to fit results with preprocessing tricks. It is, however, essential to include

978-1-5386-4359-4/18/$31.00 ©2018 IEEE

RTS as part of the active research of deep reinforcement learning algorithms as they feature long-term planning. This section outlines a thorough literature review of existing game platforms and environments and is summarized in Table I.

TABLE I
SELECTED GAME ENVIRONMENTS THAT IS ACTIVELY USED IN REINFORCEMENT LEARNING RESEARCH

Platform	RTS	Complex[1]	Year	Solved	Source
ALE	No	No	2012	Yes	[10]
Malmo Platform	No	Yes	2016	No	[11]
ViZDoom	No	Yes	2016	No	[12]
DeepMind Lab	No	Yes	2016	No	[13]
OpenAI Gym	No	No	2016	No	[14]
OpenAI Universe	No	Yes	2016	No	[15]
Stratagus	Yes	Yes	2005	No	[16]
microRTS	Yes	No	2013	No	[17]
TorchCraft	Yes	Yes	2016	No	[18]
ELF	Yes	Yes	2017	No	[19]
SC2LE	Yes	Yes	2017	No	[8]
Deep RTS	Yes	Yes	2018	No	-

A. Stratagus

Stratagus is an open source game engine that can be used to create RTS-themed games. Wargus, a clone of Warcraft II, and Stargus, a clone of StarCraft I are examples of games implemented in the Stratagus game engine. Stratagus is not an engine that targets machine learning explicitly, but several researchers have performed experiments in case-based reasoning [20], [21] and q-learning [22] using Wargus. Stratagus is still actively updated by contributions from the community.

B. Arcade Learning Environment

Bellemare *et al*. provided in 2012 the arcade learning environment that enabled researchers to conduct cutting-edge research in general deep learning [10]. The package provided hundreds of Atari 2600 environments that in 2013 allowed Minh *et al*. to do a breakthrough using Deep Q-Learning and A3C. The platform has been a critical component in several advances in RL research. [1], [3], [23]

C. microRTS

microRTS is a simple RTS game, designed to conduct AI research. The idea behind microRTS is to strip away the computational heavy game logic to increase the performance and to enable researchers to test theoretical concepts quickly [17]. The microRTS game logic is deterministic, and include options for full and partially-observable state-spaces. The primary field of research in microRTS is game-tree search techniques such as variations of Monte-Carlo tree search and minimax [17], [24], [25].

D. TorchCraft

In 2016, a research group developed TorchCraft, a bridge that enables research in the game StarCraft. TorchCraft intends to provide the reinforcement learning community with a way to allow research on complex systems where only a fraction of the state-space is available [18]. In literature, TorchCraft has been used for deep learning research [26], [27]. There is also a dataset that provides data from over 65,000 StarCraft replays [28].

[1]A Complex environment has an enormous state-space, with reward signals that are difficult to correlate to an action.

E. Malmo Platform

The Malmo project is a platform built atop of the popular game *Minecraft*. This game is set in a 3D environment where the object is to survive in a world of dangers. The paper *The Malmo Platform for Artificial Intelligence Experimentation* by Johnson *et al*. claims that the platform has all characteristics qualifying it to be a platform for general artificial intelligence research. [11]

F. ViZDoom

ViZDoom is a platform for research in visual reinforcement learning. With the paper *ViZDoom: A Doom-based AI Research Platform for Visual Reinforcement Learning* Kempka *et al*. illustrated that an RL agent could successfully learn to play the game *Doom*, a first-person shooter game, with behavior similar to humans. [29]

G. DeepMind Lab

With the paper *DeepMind Lab*, Beattie *et al*. released a platform for 3D navigation and puzzle solving tasks. The primary purpose of DeepMind Lab is to act as a platform for DRL research. [13]

H. OpenAI Gym

In 2016, Brockman *et al*. from OpenAI released GYM which they referred to as *"a toolkit for developing and comparing reinforcement learning algorithms"*. GYM provides various types of environments from following technologies: Algorithmic tasks, Atari 2600, Board games, Box2d physics engine, MuJoCo physics engine, and Text-based environments. OpenAI also hosts a website where researchers can submit their performance for comparison between algorithms. GYM is open-source and encourages researchers to add support for their environments. [14]

I. OpenAI Universe

OpenAI recently released a new learning platform called *Universe*. This environment further adds support for environments running inside VNC. It also supports running Flash games and browser applications. However, despite OpenAI's open-source policy, they do not allow researchers to add new environments to the repository. This limits the possibilities of running any environment. The OpenAI Universe is, however, a significant learning platform as it also has support for desktop games like Grand Theft Auto IV, which allow for research in autonomous driving [30].

J. ELF

The Extensive Lightweight Flexible (ELF) research platform was recently present at NIPS with the paper *ELF: An Extensive, Lightweight and Flexible Research Platform for Real-time Strategy Games*. This paper focuses on RTS game research and is the first platform officially targeting these types of games. [19]

K. StarCraft II Learning Environment

SC2LE (StarCraft II Learning Environment) is an API wrapper that facilitates access to the StarCraft II game-state using languages such as Python. The purpose is to enable reinforcement learning and machine learning algorithms to be used as AI for the game players. StarCraft II is a complex environment that requires short and long-term planning. It is difficult to observe a correlation between actions and rewards due to the imperfect state information and delayed rewards, making StarCraft II one of the hardest challenges so far in AI research.

III. Reinforcement Learning in Games

Although there are several open-source game environments suited for reinforcement learning, few of them are part of a success story. One of the reasons for this is that current state-of-the-art algorithms are seemingly unstable [30], and have difficulties to converge towards optimal policy in environments with multi-reward objectives [31]. This section exhibits the most significant achievements using reinforcement learning in games.

A. TD-Gammon

TD-Gammon is an algorithm capable of reaching an expert level of play in the board game *Backgammon* [7], [32]. The algorithm was developed by Gerald Tesauro in 1992 at IBM's Thomas J. Watson Research Center. TD-Gammon consists of a three-layer artificial neural network (ANN) and is trained using a reinforcement learning technique called *TD-Lambda*. TD-Lambda is a temporal difference learning algorithm invented by Richard S. Sutton [33]. The ANN iterates over all possible moves the player can perform and estimates the reward for that particular move. The action that yields the highest reward is then selected. TD-Gammon is the first algorithm to utilize self-play methods to improve the ANN parameters.

B. AlphaGO

In late 2015, *AlphaGO* became the first algorithm to win against a human professional Go player. AlphaGO is a reinforcement learning framework that uses Monte-Carlo tree search and two deep neural networks for value and policy estimation [9]. Value refers to the expected future reward from a state assuming that the agent plays perfectly. The policy network attempts to learn which action is best in any given board configuration. The earliest versions of AlphaGO used training data from previous games played by human professionals. In the most recent version, *AlphaGO Zero*, only self-play is used to train the AI [34]. In a recent update, AlphaGO was generalized to work for Chess and Shogi (Japanese Chess) only using 24 hours to reach a superhuman level of play [2].

C. DeepStack

DeepStack is an algorithm that can perform an expert level play in Texas Hold'em poker. This algorithm uses tree-search in conjunction with neural networks to perform sensible actions in the game [35]. DeepStack is a general-purpose algorithm that aims to solve problems with imperfect information. The DeepStack algorithm is open-source and available at https://github.com/lifrordi/DeepStack-Leduc.

D. Dota 2

DOTA 2 is a complex player versus player game where the player controls a hero unit. The game objective is to defeat the enemy heroes and destroy their base. In August 2017, OpenAI invented a reinforcement learning based AI that defeated professional players in one versus one games. The training was done by only using self-play, and the algorithm learned how to exploit game mechanics to perform well within the environment. DOTA 2 is used actively in research where the next goal is to train the AI to play in a team-game based environment.

IV. The Deep RTS Learning Environment

There is a need for new RTS game environments targeting reinforcement learning research. Few game environments have a complexity suited for current state-of-the-art research, and there is a lack of flexibility the existing solutions.

The Deep RTS game environment enables research at different difficulty levels in planning, reasoning, and control. The inspiration behind this contribution is microRTS and StarCraft II, where the goal is to create an environment that features challenges between the two. The simplest configurations of Deep RTS are deterministic and non-durative. Actions in the non-durative configuration are directly applied to the environment within the next few game frames. This makes the correlation between action and reward easier to observe. The durative configuration complicates the state-space significantly because it then becomes a temporal problem that requires long-term planning. Deep RTS supports the OpenAI Gym abstraction through the Python API and is a promising tool for reinforcement learning research.

A. Game Objective

The objective of the Deep RTS challenge is to build a base consisting of a town-hall, and then strive to expand the base using gathered resources to gain the military upper hand. Military units are used to conduct attacks where the primary goal is to demolish the base of the opponent. Players start with a worker unit. The primary objective of the worker units is to expand the base offensive, defensive and to gather natural resources found throughout the game world. Buildings can further spawn additional units that strengthen the offensive capabilities of the player. For a player to reach the terminal state, all opponent units must be destroyed.

A regular RTS game can be represented in three stages: early-game, mid-game and late-game. Early-game is the gathering and base expansion stage. The mid-game focuses on the military and economic superiority, while the late-game stage is usually a deathmatch between the players until the game ends.

TABLE II
AN OVERVIEW OF AVAILABLE SCENARIOS FOUND IN THE DEEP RTS GAME ENVIRONMENT

Scenario Name	Description	Game Length	Map Size
10x10-2-FFA	2-Player game	600-900 ticks	10x10
15x15-2-FFA	2-Player game	900-1300 ticks	15x15
21x21-2-FFA	2-Player game	2000-3000 ticks	21x21
31x31-2-FFA	2-Player game	6000-9000 ticks	31x31
31x31-4-FFA	4-Player game	8000-11k ticks	31x31
31x31-6-FFA	6-Player game	15k-20k ticks	31x31
solo-score	Score Accumulation	1200 ticks	10x10
solo-resources	Resource Harvesting	600 ticks	10x10
solo-army	Army Accumulation	1200 ticks	10x10

Because Deep RTS targets a various range of reinforcement learning tasks, there are game scenarios such as resource gathering tasks, military tasks, and defensive tasks that narrows the complexity of a full RTS game. Table II shows nine scenarios currently implemented in the Deep RTS game environment. The first six scenarios are regular RTS games with the possibility of having 6 active players in a free-for-all setting. The *solo-score* scenario features an environment where the objective is to only generate as much score as possible in shortest amount of time. *solo-resources* is a game mode that focuses on resource gathering. The agent must find a balance between base expansion and resource gathering to optimally gather as many resources as possible. *solo-army* is a scenario where the primary goal is to expand the military forces quickly and launch an attack on an idle enemy. The Deep RTS game environment enables researchers to create custom scenarios via a flexible configuration interface.

B. Game Mechanics

TABLE III
CENTRAL CONFIGURATION FLAGS FOR THE DEEP RTS GAME ENGINE

Config Name	Type	Description
instant_town_hall	Bool	Spawn Town-Hall at game start.
instant_building	Bool	Non-durative Build Mode.
instant_walking	Bool	Non-durative Walk Mode.
harvest_forever	Bool	Harvest resources automatically.
auto_attack	Bool	Automatic retaliation when being attacked.
durative	Bool	Enable durative mode.

The game mechanics of the Deep RTS are flexible and can be adjusted before a game starts. Table III shows a list of configurations currently available. An important design choice is to allow actions to affect the environment without any temporal delay. All actions are bound to a tick-timer that defaults to 10, that is, it takes 10 ticks for a unit to move one tile, 10 ticks for a unit to attack once, and 300 ticks to build buildings. The tick-timer also includes a multiplier that enables adjustments of how many ticks equals a second. For each iteration of the game-loop, the tick counter is incremented, and the tick-timers are evaluated. By using tick-timers, the game-state resembles how the StarCraft II game mechanics function while lowering the tick-timer value better resembles microRTS.

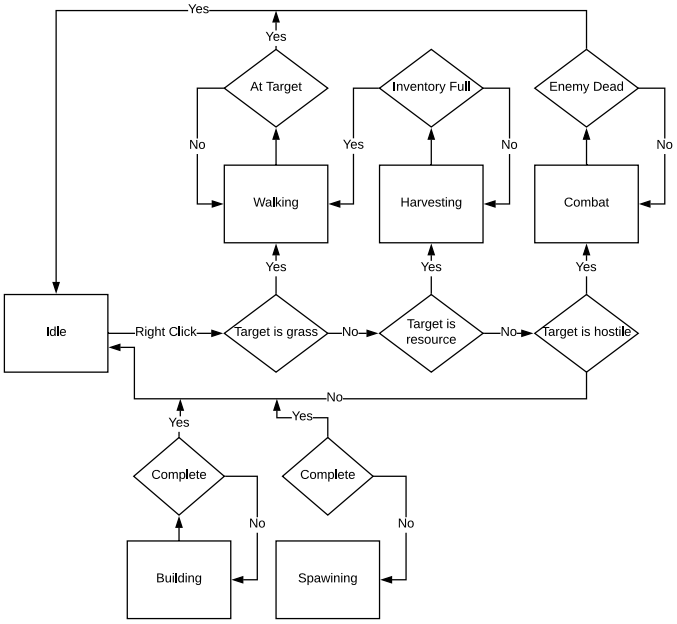


Fig. 1. Unit state evaluation based on actions and current state

All game entities (Units and Buildings) have a state-machine that determine its current state. Figure 1 illustrates a portion of the logic that is evaluated through the state-machine. Entities start in the Spawning state transitioning to the Idle state when the entity spawn process is complete. The Idle state can be considered the default state of all entities and is only transitioned from when the player interacts with the entity. This implementation enables researchers to modify the state-transitions to produce alternative game logic.

TABLE IV
THE AVAILABLE ECONOMIC RESOURCES AND LIMITS AVAILABLE TO PLAYERS IN DEEP RTS

Player Resources					
Property	**Lumber**	**Gold**	**Oil**	**Food**	**Units**
Range	0 - 10^6	0 - 10^6	0 - 10^6	0 - 6000	0 - 2000

Table IV shows the available resources and unit limits in the Deep RTS game environment. There are primarily three resources, gold, lumber, and oil that are available for workers to harvest. The value range is practically limited to the number of resources that exist on the game map. The food limit and the unit limit ensures that the player does not produce units excessively.

C. Graphics

The Deep RTS game engine features two graphical interface modes in addition to the headless mode that is used by default.

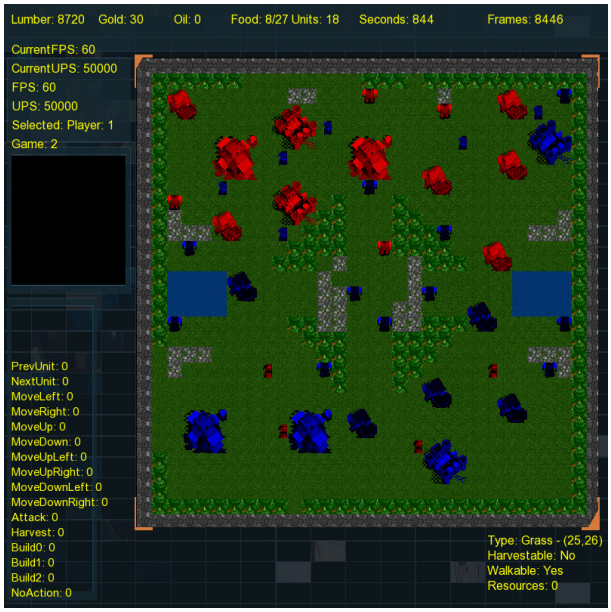


Fig. 2. Overview of a battle in the fully-observable Deep RTS state-space using the C++ graphical user interface

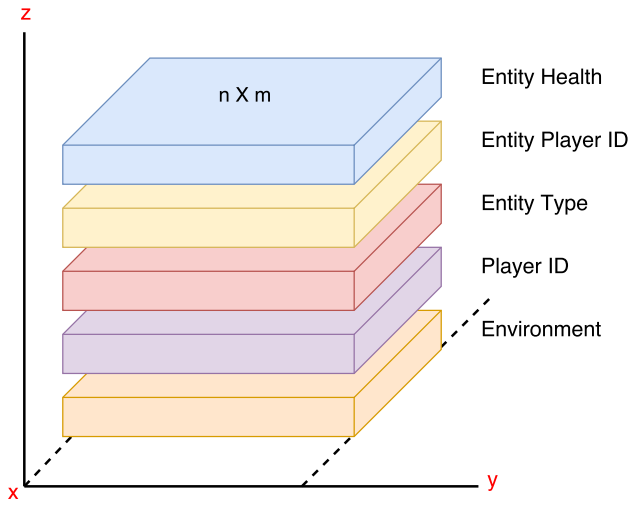


Fig. 3. Illustration of how the raw state is represented using 3-D matrices

The primary graphical interface relies on Python while the second is implemented in C++. The Python version is not interactive and can only render the raw game-state as an image. By using software rendering, the capture process of images is significantly faster because the copy between GPU and CPU is slow. The C++ implementation, seen in Figure 2 is fully interactive, enabling manual play of Deep RTS. Figure 3 shows how the raw game-state is represented as a 3-D matrix in headless mode. Deep learning methods often favor raw game-state data instead of image representation as sensory input. This is because raw data is often more concrete with clear patterns.

D. Action-space definition

The action-space of the Deep RTS game environment is separated into two abstract levels. The first level is actions that directly impact the environment, for instance, right-click, left-click, move-left, and select-unit. The next layer of abstraction is actions that combine actions from the previous layer, typically *select-unit* $\rightarrow$ *right-click* $\rightarrow$ *right-click* $\rightarrow$ *move-left*. The benefit of this abstraction is that algorithms can focus on specific areas within the game-state, and enable to build hierarchical models that each specialize in tasks (planning). The Deep RTS initially features 16 different actions in the first layer and 6 actions in the last abstraction layer, but it is trivial to add additional actions.

E. Summary

This section presents some of the central parts what the Deep RTS game environment features for reinforcement learning research. It is designed to measure the performance of algorithms accurately having a standardized API through OpenAI Gym, which is widely used in the reinforcement learning community.

V. Experiments

A. Performance considerations in Deep RTS

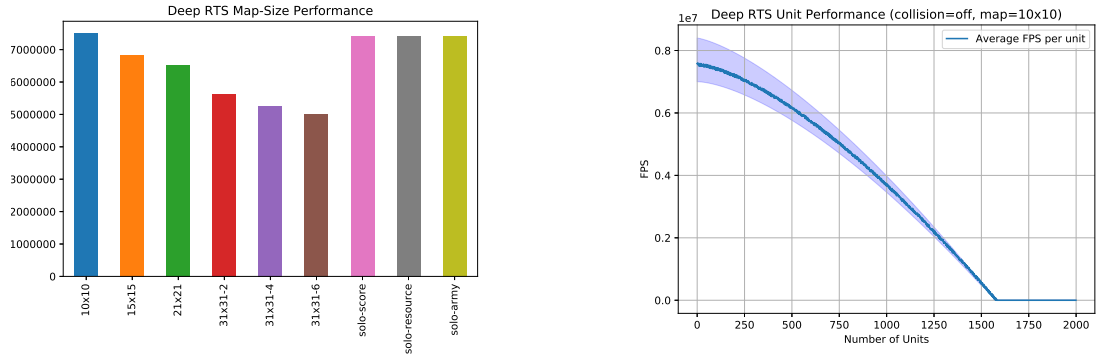
The goal of Deep RTS is to simulate RTS scenarios with ultra high-performance accurately. The performance is measured by how fast the game engine updates the game-state, and how quickly the game-state can be represented as an image. Some experiments suggest that it is beneficial to render game graphics on the CPU instead of the GPU. Because the GPU has a separate memory, there is a severe bottleneck when copying the screen buffer from the GPU to the CPU.

Figure 4a shows the correlation between the frame-rate and size of the game map. Observing the data, it is clear that the map-size has O(n) penalty to the frame-rate performance. It is vital to preserve this linearity, and optimally have the constant performance of O(1) per game update. Figure 4 extends this benchmark by testing the impact a unit has on the game performance, averaging 1 000 games for all map-sizes. The data indicates that entities have an exponential impact on the frame-rate performance. The reason for this is primarily the jump-point-search algorithm used for unit path-finding. The path-finding algorithm can be disabled using custom configurations.

The Deep RTS game environment is high-performance, with few elements that significantly reduce the frame-rate performance. While some mechanics, namely path-finding is a significant portion of the update-loop it can be deactivated by configurations to optimize the performance further.

B. Comparing Deep RTS to existing learning environments

There is a substantial difference between the performance in games targeted research and those aimed towards gaming. Table V shows that the frame-rate difference ranges from 60 to 7 000 000 for selected environments. A high frame-rate is essential because some exploration algorithms often



(a) Correlation between FPS (Y-axis) and Map-sizes (X-axis)

(b) Correlation between FPS and Number of Units

Fig. 4. FPS Performance in Deep RTS

TABLE V
COMPARISON OF THE FPS FOR SELECTED ENVIRONMENTS. THE DEEP RTS BENCHMARKS ARE PERFORMED USING MINIMUM AND MAXIMUM CONFIGURATIONS

Environment	Frame per second	Source
ALE	6,500	[10]
Malmo Platform	60-144	[11]
ViZDoom	8,300	[12]
DeepMind Lab	1,000	[13]
OpenAI Gym	60	[14]
OpenAI Universe	60	[15]
Stratagus	60-144	[16]
microRTS	11,500	[17]
TorchCraft	2,500	[18]
ELF	36,000	[19]
SC2LE	60-144	[8]
Deep RTS	24,000, 7,000,000	-

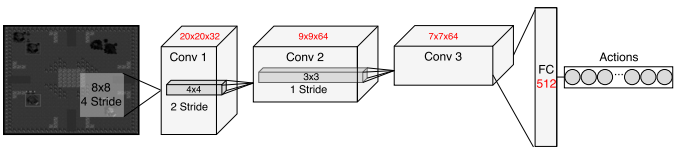


Fig. 5. Overview of the Deep Q-Network architecture used in the experiments. Inspired by the work seen in [1]

require a quick assessment of future states through forward-search. Table V shows that microRTS, ELF, and Deep RTS are superior in performance compared to other game environments. Deep RTS is measured using the largest available map (Table II) having a unit limit of 20 per player. This yields the performance of 24 000 updates-per-second. The Deep RTS game engine can also render the game state with up to 7 000 000 updates-per-second using the minimal configuration. This is a tremendous improvement on previous work and could enable algorithms with a limited time budget to do deeper tree-searches.

C. Using Deep Q-Learning in Deep RTS

At the most basic level, Q-Learning utilizes a table for storing $(s, a, r, s^{'})$ pairs, where s is the states, a is the actions, r the rewards,and $s^{'}$ the next state. Instead, a non-linear function approximation can be used to approximate $Q(s, a; \theta)$. This is called **Deep-Q Learning**. θ describes the tunable parameters (weights) for the approximation function. Artificial neural networks are used as an approximation function for the Q-Table but at the cost of stability [3]. Using artificial neural networks is much like compression found in JPEG images. The compression is *lossy*, and some information is lost during the compression. Deep Q-Learning is thus unstable, since values may be incorrectly encoded during training [36].

This paper presents experimental results using the Deep Q-Learning architecture from [3], [37]. Figure 5 shows the network model, and figure 6 illustrates the averaged training loss of 100 agents. The agent uses gray-scale image game-state representations with an additional convolutional layer to decrease the training time, but can also achieve comparable results after approximately 800 episodes of training with the exact architecture from [3][2]. The graph shows that the agent quickly learns the correlation between game-state, action and the reward function. The loss quickly stabilizes at a relatively low value, but it is likely that very small optimizations in the parameters have a significant impact on the agent's performance.

Figure 7a shows the win-rate against an AI with a random-play strategy. The agent quickly learns how to perform better than random behavior, and achieves 70 % win-rate at episode 1 250. Figure 7b illustrates the same agent playing against a rule-based strategy. The graph shows that the Deep Q-Network can achieve an average of 50 % win-rate over a 1 000 games. This strategy is considered an easy to moderate player, where its strategy is to expand the base towards the opponent and build a military force after approximately 600 seconds. Figure 2 shows how the rule-based player (blue) expands the base to gain the upper hand.

The experimental results presented in this paper show that the Deep RTS game environment can be used to train deep

[2]Each episode contains approximately 1 000 epochs of training with a batch size of 16

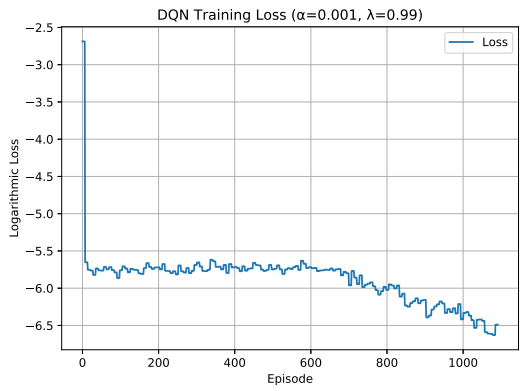


Fig. 6. Training loss of the Deep Q-Network. Each episode consists of approximately 1 000 epochs.

reinforcement learning algorithms. The Deep Q-Network does not achieve super-human expertise but performs similarly to a player of easy to moderate skill level, which is a good step towards a high-level AI.

VI. Conclusion and Future Work

This paper is a contribution towards the continuation of research into deep reinforcement learning for RTS games. The paper summarizes previous work and outlines the few but essential success stories in reinforcement learning. The Deep RTS game environment is a high-performance RTS simulator that enables rapid research and testing of novel reinforcement learning techniques. It successfully fills the gap between the vital game simulator microRTS, and StarCraft II, which is the ultimate goal for reinforcement learning research for the RTS game genre.

The hope is that Deep RTS can bring insightful results to the complex problems of RTS [17] and that it can be a useful tool in future research.

Although the Deep RTS game environment is ready for use, several improvements can be applied to the environment. The following items are scheduled for implementation in the continuation of Deep RTS:

- Enable LUA developers to use Deep RTS through LUA bindings.
- Implement a generic interface for custom graphics rendering.
- Implement duplex WebSockets and ZeroMQ to enable any language to interact with Deep RTS
- Implement alternative path-finding algorithms to increase performance for some scenarios
- Add possibility for memory-based fog-of-war to better mimic StarCraft II

VII. Acknowledgements

We would like to thank Santiago Ontanon at the University of Drexel for his excellent work on microRTS. microRTS has to us been a valuable tool for benchmarks in the reinforcement learning domain and continues to be the goto environment for research in tree-search algorithms.

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning," *arxiv preprint arXiv:1312.5602*, dec 2013. [Online]. Available: http://arxiv.org/abs/1312.5602

[2] W. Gaetz, S. K. Kessler, T. P. Roberts, J. I. Berman, T. J. Levy, M. Hsia, D. Humpl, E. S. Schwartz, S. Amaral, B. Chang, and L. S. Levin, "Massive cortical reorganization is reversible following bilateral transplants of the hands: evidence from the first successful bilateral pediatric hand transplant patient," *Annals of Clinical and Translational Neurology*, vol. 5, no. 1, pp. 92–97, dec 2018. [Online]. Available: https://arxiv.org/abs/1712.01815

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, feb 2015. [Online]. Available: http://www.nature.com/articles/nature14236

[4] P. Mirowski, R. Pascanu, F. Viola, H. Soyer, A. J. Ballard, A. Banino, M. Denil, R. Goroshin, L. Sifre, K. Kavukcuoglu, D. Kumaran, and R. Hadsell, "Learning to Navigate in Complex Environments," *arXiv preprint arXiv:1611.03673*, nov 2016. [Online]. Available: http://arxiv.org/abs/1611.03673

[5] M. A. Yasin, W. A. Al-Ashwal, A. M. Shire, S. A. Hamzah, and K. N. Ramli, "Tri-band planar inverted F-antenna (PIFA) for GSM bands and bluetooth applications," *ARPN Journal of Engineering and Applied Sciences*, vol. 10, no. 19, pp. 8740–8744, 2015.

[6] G. Konidaris and A. Barto, "Autonomous shaping," *Proceedings of the 23rd international conference on Machine learning - ICML '06*, pp. 489–496, 2006. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1143844.1143906

[7] G. Tesauro, "TD-Gammon, a Self-Teaching Backgammon Program, Achieves Master-Level Play," *Neural Computation*, vol. 6, no. 2, pp. 215–219, 1994. [Online]. Available: http://www.mitpressjournals.org/doi/10.1162/neco.1994.6.2.215

[8] O. Vinyals, T. Ewalds, S. Bartunov, P. Georgiev, A. S. Vezhnevets, M. Yeo, A. Makhzani, H. Küttler, J. Agapiou, J. Schrittwieser, J. Quan, S. Gaffney, S. Petersen, K. Simonyan, T. Schaul, H. van Hasselt, D. Silver, T. Lillicrap, K. Calderone, P. Keet, A. Brunasso, D. Lawrence, A. Ekermo, J. Repp, and R. Tsing, "StarCraft II: A New Challenge for Reinforcement Learning," *Proceedings, The Thirteenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, pp. 86–92, 2017. [Online]. Available: http://arxiv.org/abs/1708.04782

[9] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

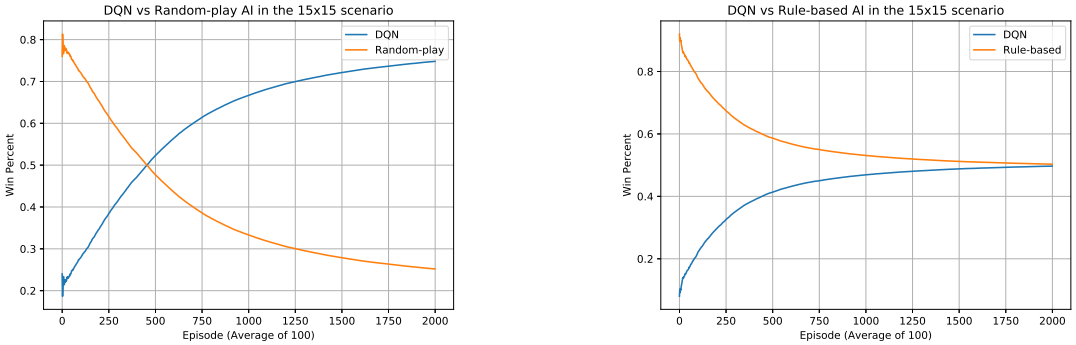
[10] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *IJCAI International Joint Conference on Artificial Intelligence*, vol. 2015-Janua, pp. 4148–4152, 2015. [Online]. Available: http://arxiv.org/abs/1207.4708

[11] M. Johnson, K. Hofmann, T. Hutton, and D. Bignell, "The malmo platform for artificial intelligence experimentation," *IJCAI International Joint Conference on Artificial Intelligence*, vol. 2016-Janua, pp. 4246–4247, 2016. [Online]. Available: https://www.ijcai.org/Proceedings/16/Papers/643.pdf

[12] E. Perot, M. Jaritz, M. Toromanoff, and R. D. Charette, "End-to-End Driving in a Realistic Racing Game with Deep Reinforcement Learning," *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, vol. 2017-July, pp. 474–475, may 2017. [Online]. Available: http://arxiv.org/abs/1605.02097

[13] C. Beattie, J. Z. Leibo, D. Teplyashin, T. Ward, M. Wainwright, H. Küttler, A. Lefrancq, S. Green, V. Valdés, A. Sadik, J. Schrittwieser, K. Anderson, S. York, M. Cant, A. Cain, A. Bolton, S. Gaffney, H. King, D. Hassabis, S. Legg, and S. Petersen, "DeepMind Lab," *arXiv preprint arXiv:1612.03801*, dec 2016. [Online]. Available: http://arxiv.org/abs/1612.03801

[14] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI Gym,"



(a) DQN vs Random-play AI in the 15x15-2-FFA scenario

(b) DQN vs Rule-based AI in the 15x15-2-FFA scenario

Fig. 7. Performance comparison of agents using Deep Q-Network, random-play, and rule-based strategies

arXiv preprint arXiv:1606.01540, jun 2016. [Online]. Available: http://arxiv.org/abs/1606.01540

[15] OpenAI, "OpenAI Universe," 2017. [Online]. Available: https://universe.openai.com/

[16] M. Ponsen, S. Lee-Urban, H. Muñoz-Avila, D. Aha, and M. Molineaux, "Stratagus: An open-source game engine for research in real-time strategy games," *Reasoning, Representation, and Learning in Computer Games*, no. Code 5515, p. 78, 2005. [Online]. Available: https://pdfs.semanticscholar.org/d005/87160d3b37c70f238ebd92c71454479e829e.pdf

[17] S. Ontanon, "The combinatorial multi-armed bandit problem and its application to real-time strategy games," in *Ninth Artificial Intelligence and Interactive Digital ...*, 2013, pp. 58–64. [Online]. Available: http://www.aaai.org/ocs/index.php/AIIDE/AIIDE13/paper/viewPaper/7377

[18] G. Synnaeve, N. Nardelli, A. Auvolat, S. Chintala, T. Lacroix, Z. Lin, F. Richoux, and N. Usunier, "TorchCraft: a Library for Machine Learning Research on Real-Time Strategy Games," *arxiv preprint arXiv:1611.00625*, nov 2016. [Online]. Available: http://arxiv.org/abs/1611.00625

[19] Y. Tian, Q. Gong, W. Shang, Y. Wu, and C. L. Zitnick, "ELF: An Extensive, Lightweight and Flexible Research Platform for Real-time Strategy Games," *Advances in Neural Information Processing Systems*, pp. 2656–2666, jul 2017. [Online]. Available: http://arxiv.org/abs/1707.01067

[20] S. Ontañón, K. Mishra, N. Sugandh, and A. Ram, "Learning from demonstration and case-based planning for real-time strategy games," in *Studies in Fuzziness and Soft Computing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, vol. 226, pp. 293–310. [Online]. Available: http://link.springer.com/10.1007/978-3-540-77465-5{_}15

[21] I. Fathy, M. Aref, O. Enayet, and A. Al-Ogail, "Intelligent online case-based planning agent model for real-time strategy games," in *Proceedings of the 2010 10th International Conference on Intelligent Systems Design and Applications, ISDA'10*. IEEE, nov 2010, pp. 445–450. [Online]. Available: http://ieeexplore.ieee.org/document/5687225/

[22] U. Jaidee and H. Muñoz-Avila, "CLASSQ-L: A Q-Learning Algorithm for Adversarial Real-Time Strategy Games," pp. 8–13, 2012. [Online]. Available: http://www.aaai.org/ocs/index.php/AIIDE/AIIDE12/paper/viewFile/5515/5734

[23] B. Lindström, I. Selbing, T. Molapour, and A. Olsson, "Racial Bias Shapes Social Reinforcement Learning," *Psychological Science*, vol. 25, no. 3, pp. 711–719, feb 2014. [Online]. Available: http://arxiv.org/abs/1602.01783

[24] N. A. Barriga, M. Stanescu, and M. Buro, "Game Tree Search Based on Non-Deterministic Action Scripts in Real-Time Strategy Games," *IEEE Transactions on Computational Intelligence and AI in Games*, pp. 1–1, 2017. [Online]. Available: http://ieeexplore.ieee.org/document/7954767/

[25] A. Shleyfman, A. Komenda, and C. Domshlak, "On combinatorial actions and CMABs with linear side information," in *Frontiers in Artificial Intelligence and Applications*, vol. 263, 2014, pp. 825–830.

[26] D. Churchill, Z. Lin, and G. Synnaeve, "An Analysis of Model-Based Heuristic Search Techniques for StarCraft Combat Scenarios," pp. 8–14, 22017. [Online]. Available: https://aaai.org/ocs/index.php/AIIDE/AIIDE17/paper/view/15916

[27] P. Peng, Y. Wen, Y. Yang, Q. Yuan, Z. Tang, H. Long, and J. Wang, "Multiagent Bidirectionally-Coordinated Nets: Emergence of Human-level Coordination in Learning to Play StarCraft Combat Games," *arxiv preprint arXiv:1703.10069*, mar 2017. [Online]. Available: http://arxiv.org/abs/1703.10069

[28] Z. Lin, J. Gehring, V. Khalidov, and G. Synnaeve, "STARDATA: A StarCraft AI Research Dataset," *Proceedings, The Thirteenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, aug 2017. [Online]. Available: http://arxiv.org/abs/1708.02139

[29] E. Perot, M. Jaritz, M. Toromanoff, and R. D. Charette, "End-to-End Driving in a Realistic Racing Game with Deep Reinforcement Learning," *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, vol. 2017-July, pp. 474–475, may 2017. [Online]. Available: https://arxiv.org/abs/1605.02097

[30] Y. Li, "Deep Reinforcement Learning: An Overview," *arXiv preprint arXiv:1701.07274*, pp. 1–30, 2017. [Online]. Available: http://arxiv.org/abs/1701.07274

[31] T. Mannucci, E. J. van Kampen, C. de Visser, and Q. Chu, "Safe Exploration Algorithms for Reinforcement Learning Controllers," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 16, pp. 1437–1480, 2017. [Online]. Available: http://www.jmlr.org/papers/volume16/garcia15a/garcia15a.pdf

[32] G. Tesauro, "Temporal difference learning and TD-Gammon," *Communications of the ACM*, vol. 38, no. 3, pp. 58–68, 1995. [Online]. Available: http://portal.acm.org/citation.cfm?doid=203330.203343

[33] R. S. Sutton and A. G. Barto, "Chapter 12: Introductions," *Acta Physiologica Scandinavica*, vol. 48, no. Mowrer 1960, pp. 57–63, 1960.

[34] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. Van Den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[35] M. Moravčík, M. Schmid, N. Burch, V. Lisý, D. Morrill, N. Bard, T. Davis, K. Waugh, M. Johanson, and M. Bowling, "DeepStack: Expert-level artificial intelligence in heads-up no-limit poker," *Science*, vol. 356, no. 6337, pp. 508–513, jan 2017. [Online]. Available: http://arxiv.org/abs/1701.01724http://dx.doi.org/10.1126/science.aam6960

[36] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, vol. abs/1509.0, sep 2015. [Online]. Available: http://arxiv.org/abs/1509.02971

[37] W. Chen, M. Zhang, Y. Zhang, and X. Duan, "Exploiting meta features for dependency parsing and part-of-speech tagging," *Artificial Intelligence*, vol. 230, pp. 173–191, sep 2016. [Online]. Available: http://arxiv.org/abs/1509.06461

New And Surprising Ways to Be Mean

Christian Guckelsberger
Department of Computing
Goldsmiths, University of London
London, UK
c.guckelsberger@gold.ac.uk

Christoph Salge
School of Computer Science
University of Hertfordshire
Hatfield, UK
c.salge@herts.ac.uk

Julian Togelius
Tandon School of Engineering
New York University
Brooklyn, USA
julian.togelius@nyu.edu

***Abstract*—Creating Non-Player Characters (NPCs) that can react robustly to unforeseen player behaviour or novel game content is difficult and time-consuming. This hinders the design of believable characters, and the inclusion of NPCs in games that rely heavily on procedural content generation. We have previously addressed this challenge by means of empowerment, a model of intrinsic motivation, and demonstrated how a coupled empowerment maximisation (CEM) policy can yield generic, companion-like behaviour. In this paper, we extend the CEM framework with a minimisation policy to give rise to adversarial behaviour. We conduct a qualitative, exploratory study in a dungeon-crawler game, demonstrating that CEM can exploit the affordances of different content facets in adaptive adversarial behaviour without modifications to the policy. Changes to the level design, underlying mechanics and our character's actions do not threaten our NPC's robustness, but yield new and surprising ways to be mean.**

***Index Terms*—Non-Player Characters, Adversaries, Intrinsic Motivation, Coupled Empowerment Maximisation**

I. Introduction

Non-Player Characters (NPCs) in video games serve many purposes: they can be quest givers, conversation partners, leaders, sidekicks or other kinds of collaborators [1]. But in many cases they are *adversaries*. Adversarial NPCs also come in many forms, their behaviour varying according to the game genre, the design affordances, and the underlying algorithms. Treanor et al. [2] make the fundamental distinction between *AI as Adversary* and *AI as Villain*. Adversaries are designed to defeat the player without resorting to cheating, e.g. an AI for Chess or Go. The objective of an NPC villain in contrast is not to defeat the player but to create an interesting challenge which can be overcome eventually. We refer to both types simply as adversaries.

Irrespective of the type, an NPC's primary goal is usually to convey a special player experience. A substantial part of this experience is shaped by the believability of their behaviour [1]: a believable adversary can, amongst others, adapt to changes in the world and allows the player to attribute goal-ownership. In existing adversary AI however, these attributes are either not present, or very shallow.

NPCs in video games are largely hand-authored, using representations such as finite state machines and behaviour trees. This limits most NPC AI to a particular game and a specific role. While such NPCs might appear to own their goals, they will hardly adapt to unanticipated player behaviour or changes in the game world. The latter aspect is partly alleviated by reinforcement learning, evolutionary approaches or planning. However, there is two caveats. Algorithms such as Monte Carlo Tree Search (MCTS) are typically targeted towards maximising adversarial efficacy against the player, resulting in blunt and single-faceted behaviour. Procedural personas [3] contribute to the impression of more multi-faceted behaviour by optimising a set of pre-specified utilities. However, even these advanced approaches usually rely on objective functions, rewards and training samples which are strongly tied to specific affordances of the game world. As soon as this world changes, the basis for their behaviour and thus their believability is lost.

An alternative approach is to use models of intrinsic motivation [4] to drive NPC behaviour. Models of intrinsic motivation do not rely on externally specified rewards, and thus allow an agent to act sensibly even if its means to interact with the world change. As intrinsic motivations are usually aligned with key drivers of agency, this can give the appearance of goal-directness. Merrick and Maher [5] have used curiosity and learning progress as intrinsic reward signals in reinforcement learning to drive NPC behaviour. However, their NPCs work in isolation, and interactions with the player would be incidental and likely shallow.

In this paper, we address the challenge of creating *generic adversarial NPCs*, i.e. NPCs that can adapt and respond to substantial changes in the game environment, mechanics and a character's abilities, and that exhibit a wide range of new and surprising adversarial behaviours that are not uniquely focused on winning over the player. We prose to use the intrinsic motivation formalism of *Coupled Empowerment Minimisation* (CEM) [6], an action policy based on the information-theoretic quantity *empowerment* [7]. Empowerment quantifies the options available to an agent in terms of availability and visibility. In the stochastic case, it generalises to an agent's potential and perceivable influence on the game world, including other agents such as the player. Empowerment forms the basis of *empowerment maximisation*, an action policy which drives agents towards states where they have a higher influence on their environment. CEM is an extension of this principle to the multi-agent case. The main idea behind CEM is that an agent not only maximises its own- but also maximises or minimises the empowerment of one or more other characters. In previous work [6], we have exploited the *maximisation case* to formalise companion-like behaviour in a very general and flexible way. We expect the policy to yield sensible NPC behaviour in any game where a player's progress towards a goal is accompanied by an increase in options and influence, and thus empowerment.

978-1-5386-4359-4/18/$31.00 ©2018 IEEE

This is the case for most games: consider e.g. the effect of accumulating resources and building units in strategy games, collecting inventory items in an RPG, or using power-ups or additional and stronger weapons in a shooter.

In this work, we look at the *minimisation* case to design more believable adversarial NPCs. Our NPCs essentially choose actions which increase their own-, and decrease the player's empowerment. Note that this is different from simply maximising or minimising a utility such as score or health, and promises to give rise to highly adaptive, unexpected and novel adversarial behaviour. We explore CEM to drive adversarial NPC behaviour in different levels of a turn- and tile-based dungeon-crawler game, where an CEM-driven agent is confronted with changes in the environment and its own abilities. A qualitative analysis demonstrates that CEM yields sensible and interesting adversarial behaviour across a range of game modes. Relating to Treanor et al. [2], we show how different parametrisations of our policy give rise to different adversary types, from opportunists to super-villains.

II. Coupled Empowerment Maximisation

In a nutshell, a CEM-driven agent acts to maximise its own-, while either maximising or minimising another agent's empowerment. We investigate the minimisation case here, and therefore complement previous work [6]. CEM relies on two types of empowerment: *(vanilla) empowerment* as briefly mentioned in the introduction, and the distinct *transfer-empowerment*. We now introduce both quantities formally. Our focus is on games that are discrete in time and space. However, continuous empowerment implementations exist. An extensive survey of motivations, intuitions and past research on empowerment can be found in [8]. CEM has previously been covered in [9], [6] and [10].

A. Empowerment and Transfer Empowerment

Empowerment is an information-theoretic quantity. It is zero when the agent has no control over what it can perceive, i.e. when all actions lead to the same or a random sensor state, and it increases when different actions lead to separate perceivable outcomes. We represent an agent's actions, its future sensor states, and the state of the environment as random variables A, S, and R, respectively. The causal conditional probability distribution $p(S_{t+1}|A_t, r_t)$ then models the impact of the agent's actions, performed in a specific environment state $R_t = r_t$, on its future sensor states. For the calculation of empowerment, this distribution is interpreted as a memoryless, potentially noisy information-theoretic communication channel.

Vanilla empowerment $\mathfrak{E}_{r_t}$ in a given environment state r_t is calculated as the *channel capacity*, corresponding to the maximum potential information flow that an agent could induce into its future sensor state by a suitable choice of actions. More generally, we consider a sequence of actions $A_t^n = (A_t, \dots, A_{t+n-1})$ corresponding to a lookahead of n. With *n-step empowerment* we then measure an agent's influence on its future sensor state n steps in the future:

$$\mathfrak{E}^n_{r_t} = \max_{p(a_t^n)} I(S_{t+n}; A_t^n | r_t) \tag{1}$$

The term being maximised represents the mutual information between the actuator and future sensor states, given the current environment state r_t.

Transfer empowerment $\mathfrak{E}^{T,n}_{r_t}$ relates the actions and sensor of *two* agents: it quantifies the potential influence the active agent's actions have on the other's future sensor state. The channel capacity underlying both empowerment types can be exactly determined using the Blahut-Arimoto algorithm [11], [12]. For an introduction to the information-theoretic notions see [13], and [8] for a detailed introduction to empowerment.

B. Coupled Empowerment Maximisation

CEM is an extension of empowerment maximisation to the multi-agent case, and we consequently have to account for the actions of other agents. In this paper, we focus on the turn-wise interaction of *one* NPC with the player. Each interaction cycle is initiated by the player performing an action, which the NPC reacts to in the next time step. Both agents can affect the other either explicitly, or implicitly through their impact on the shared game world. We hypothesise that decreasing the player's empowerment gives rise to adversarial behaviour. To test this hypothesis, we model the NPC's policy such that it not only maximises the NPC's own, but also minimises the player's empowerment:

$$\pi(r_t) = \arg\max_{a_t} \left(\alpha_A \cdot \mathbb{E}[\mathfrak{E}^{A,n}_{r_{t+2}}]_{a_t} + \alpha_P \cdot \mathbb{E}[\mathfrak{E}^{P,n}_{r_{t+1}}]_{a_t} + \alpha_T \cdot \mathbb{E}[\mathfrak{E}^{T,n}_{r_{t+2}}]_{a_t} \right) \tag{2}$$

Here, parameters α_A, α_P and α_T weight the expected adversary, player and transfer n-step empowerment in the overall coupling. The *adversary-player transfer empowerment* serves the maintenance of *operational proximity*: even if the NPC cannot affect the player's empowerment at the current point in time, it will try to remain in states where it can at least affect the player's perception, and thus increase the likelihood of affecting its empowerment in the future.

Determining the optimal action is a two-stage process. For the first stage, we have to note that empowerment is a state dependent quantity and the policy thus involves expectations over the NPC's actions. For the computation of these policies, the NPC first determines which environment states its own actions could yield at $t + 1$. This is where the player acts next, and where player empowerment will be computed. For the calculation of its own and adversary-player transfer empowerment however, it needs to anticipate the consequences of the player's actions on the distribution of environment states at $t + 2$. In the second stage, the NPC then calculates player empowerment in $t + 1$, as well as its own and transfer empowerment in $t + 2$. This requires another $2n$ rounds of anticipation steps. For the calculation of empowerment, the resulting environment states are transformed to potentially limited sensor states S.

Unlike algorithms such as Minimax or MCTS, states are uniformly expanded up to a fixed depth, and only distin-

guished in terms of whether they are perceived differently. Here, we model the player's policy as a uniform distribution.

C. Health-Performance Consistency

In many games, a decrease in health or a different core game quantity does not necessarily result in the decline of a character's abilities. For short lookaheads n, empowerment would thus remain unaffected. We counteract this by adopting the transformation of *health-performance consistency* (HPC) from previous work [6]. It reduces the probability of a character's action to lead into the follow-up state originally prescribed by the environment dynamics proportionally to its remaining health. The more an agent's health decreases, the more likely its actions will be ineffective. Formally:

$$p(r_{t+1}|a_t, r_t) = \begin{cases} 1 - \gamma + \gamma p(r_{t+1}|a_t, r_t), & \text{if } r_{t+1} = r_t \\ \gamma p(r_{t+1}|a_t, r_t) & \text{else.} \end{cases}$$

Here, $\gamma = h_t/h_{max}$, the ratio of the agent's remaining and maximum health. This can of course be modified to model non-linear changes. We use HPC as a means for optimisation, but it is not a necessity: given a large enough lookahead n, the long-term effect of decreasing an agent's health will be reflected in its empowerment.

III. Evaluation

We hypothesise that CEM yields highly adaptive, adversarial behaviour. The behavioural dynamics following changes to the game might be surprising even for the game's designers, and likely increase the believability of our characters. However, quantitative means to evaluate gameplay and player experience provide insufficient evidence, as they cannot capture these dynamics as well as novelty and believability in sufficient detail. We consequently perform a qualitative, exploratory study, providing the necessary insights for a quantitative study to follow in future work. This evaluation is complemented with online videos of the NPC behaviour.

A. Method

We conduct three individual experiments to test our hypothesis and investigate how a CEM-driven NPC copes with increasingly tough challenges in game development and research: (1) predator-and-prey behaviour as present in many games, (2) the exploitation of affordances in the agent-environment interaction, and (3) the ability to interact with the player from a distance. The latter is controlled by the experimenters. To probe how CEM contributes to novelty and adaptivity, we change the environment dynamics and the abilities of both characters and analyse the emergent behaviour. Each experiment comprises a number of scenarios.

Due to the richness of our testbed, an exhaustive search through the space of environmental features and character abilities would be infeasible. We consequently focus on those combinations that yield the biggest difference in emergent behaviour. The CEM parameters, i.e. the weights α and the agent's lookahead n, cannot be evaluated exhaustively either in a qualitative study. Instead, we highlight how specific configurations allows us to model different adversary types, thus stressing the opportunities in parameter fine-tuning.

B. Testbed

We have adopted our dungeon-crawler testbed from previous work on CEM-driven general companion NPCs [6] with the goal to support comparisons and provide a basis for a future joint quantitative evaluation. The testbed is discrete in time and action/state space, which simplifies the computation of our policy and the analysis of behaviour.

The game is populated by the player character and one CEM-driven NPC. Characters interact in turn-wise order, and the player must navigate to a goal-tile to win the game. All characters have a current and maximum amount of health points, which are indicated by numbers at the bottom of their avatars. To provide rich challenges for adaptation, we have extended our previous testbed substantially with both new environmental features and character abilities. Tbl. I and II provide an overview of the various features/abilities, their dynamics and the rationale behind their inclusion.

The sensors of player and adversary are *asymmetric*, *local* and *non-overlapping*. They are asymmetric in that the player can also perceive the game status, while the NPC cannot. Locality means that e.g. the NPC only perceives the player or other dynamic game elements within a fixed radius. This determines a character's perceptive field, which is only constrained by walls. Other characters within that field are sensed by their id and relative position. In addition to its own position, sensors only comprise the agent's own rotation and health, but do not allow introspection into other characters. This separation is crucial to avoid overlap between empowerment types.

We assume a *default configuration* of our agents which is adapted in the experiments. In this minimal setup, characters can only idle and move. Their sensor is limited to a three-cell radius. Furthermore, they are initialised with two of two health points ($h_t, h_{max} = 2$). This allows them to take damage without dying right away, and to make use of rechargers. We compute empowerment for a 3-step lookahead (n=3), and assume an initial weighting of α_A=.5, α_P=−.5 and α_T=.1. In this initial setup, the CEM-driven NPC bases its decision-making on the maximisation of its own, and on the minimisation of the player's empowerment to the same extent. We later deviate from this equilibrium and show how unbalanced configurations yield radically different behaviours. Parameters are then chosen from $\alpha_A, \alpha_T \in [0, 1]$ and $\alpha_P \in [-1, 0]$. In our experiments, actions are always chosen greedily with respect to the policy. We only report on these settings if they deviate from the default configuration.

C. Experiment 1: Predator-and-Prey

The goal of the first experiment is to illustrate the different forces within CEM, and to highlight the policy's potential to give rise to the classic predator-and-prey behaviour which is quintessential to many games. Fig. 1a shows the initial state of the environment, consisting of an arena surrounded by walls, divided by a wall with small spaces on the sides to

TABLE I: Level elements in dungeon-crawler testbed

Sprite	Type	Dynamics	Reason for inclusion
	Goal	Once the character moves on this tile, the game is won.	To provide a gradient for progression within a level.
	Wall	Immovable obstacle. Cannot be penetrated by attacks, and hides anything behind from character's perception.	To structure level and provide choke points for specific interactions. Allows for discovery of hidden elements.
	Lava	Decreases a character's health by a fixed amount for each time step it remains on the field.	Structures environment further and introduces health trade-offs. Allows for rich interaction with pushing.
	Recharger	Increases a character's health by a fixed amount for each time step it remains on the field.	Makes health a manageable and expendable resource that might be traded for other gains.
	Turret	Shoots arrow, inflicting a fixed health damage on the first character being hit. Here pointing east.	Serves as threat separate from characters' attack facilities, and bears danger of self-inflicted damage.
	Trigger	Activates connected turret for each time step that a character remains on the tile.	Triggers can be far off the activated turret and thus allow to strike remotely.

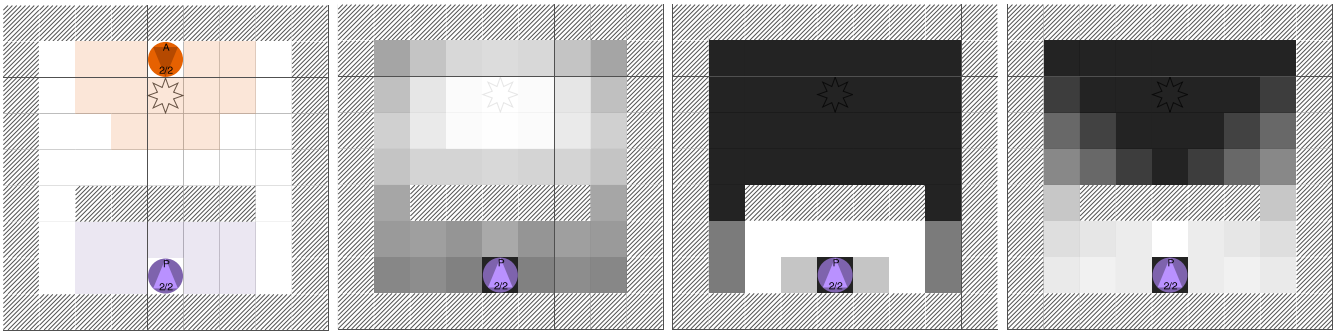
TABLE II: Character abilities in dungeon-crawler testbed

Action	Dynamics	Reason for inclusion
Idle	Causes no change to the current game state.	Represents fallback if other actions are disadvantageous.
Move	Move non-diagonally into adjacent cell if there is no obstacle. Otherwise only changes character orientation.	Common mechanic allowing for exploration, hiding and change of position as reaction to other characters.
Push	In addition to moving, shift adjacent characters in the movement direction if there is no obstruction.	Allows for complex interactions with the environment by pushing others into lava, rechargers, or a turret's target range.
Fly	Allows to move over lava fields without taking damage. The character can still benefit from rechargers.	A way to access previously inaccessible parts of a level, and make other characters face new obstacles.
Melee attack	Causes damage to adjacent characters if being faced. The amount of health damage is predefined.	Common mechanic for predator-and-prey scenarios. Requires to run away or attack from a distance before others close.
Range attack	Reduces health of first character in current direction within attack range. Damage and range are predefined.	Allows to imbalance attack options based on spatial proximity, making seeking cover a sensible move to escape damage.
Heal	Increases health of adjacent, faced character by fixed amount and up to maximum health for that character.	To check if an action which conventionally does not feature in player-adversary interaction is chosen and to what effect.

pass through. The adversary ('A', orange) is at the top and faces south, while the player ('P', purple) is situated at the bottom and faces north. Their perceptive field is shown in orange and purple, respectively.

We have made this environment deliberately simple to familiarise the reader with the different empowerment types in the CEM policy. Fig. 1b shows the adversary NPC's empowerment for a 3-step lookahead. Each hue indicates the agent's empowerment if it was moved to that position, but the player's position remained the same. Brighter hues represent higher empowerment. In the default configuration, the agents can only move or idle, and empowerment is consequently very sensitive to degrees of freedom in movement: it is lower where the agent would be blocked, e.g. close to walls and corners. The choke point between the middle and side walls has particularly low empowerment, separating the lower and upper parts of the environment into distinct gradients with local maxima. The player's 3-step empowerment is very similar, given that both agents by default possess the same abilities. Fig. 1c illustrates the transfer empowerment from the adversary to the player, for different positions of the adversary. Recall that this empowerment type corresponds to the influence the NPC has on the player's sensor. Hence, for $n{=}1$, it is only non-zero within the player's perceptive field. For larger lookaheads in contrast, it fades out to states from which the NPC could influence the player's perception with some n-step action sequences (Fig. 1d). This demonstrates that transfer empowerment does not measure perceptibility, but operational-, or in this case, spatial proximity.

The contrast between adversary- and adversary-player transfer empowerment highlights how the different empowerment types compete in the CEM policy: If the adversary NPC only considered transfer empowerment, it would move closer to the player; maximising its own empowerment however would require to stay in the middle of the upper part and avoid the choke points on the sides. This trade-off is mediated by the α-weights, which can be used to design for different behaviours. Consider the following example: If we equip our adversary NPC with the ability to perform range attacks but stick to the default parameter setup, it remains in the upper area. Nonetheless, if the player moves into this territory, it is killed with two directed shots. We classify this type of adversary as *opportunist*. In contrast, if we increase the negative weight of the player's- while decreasing the



(a) Initial state (b) $\mathfrak{E}^{A,3}$ (c) $\mathfrak{E}^{T,1}$ (d) $\mathfrak{E}^{T,3}$

Fig. 1: Experiment 1. Initial state with perceptive field of adversary and player, followed by adversary ($\mathfrak{E}^{A,n}$) and adversary-player transfer empowerment ($\mathfrak{E}^{T,n}$), the latter for lookaheads $n = 1, 3$. Brighter hues indicate higher empowerment.

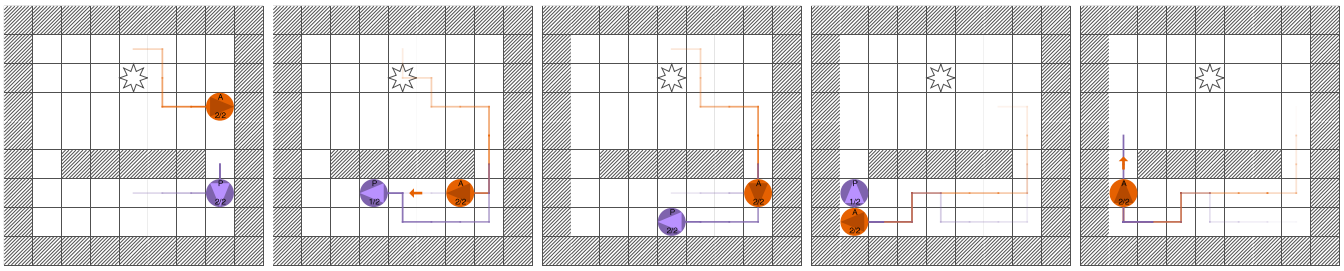


Fig. 2: Experiment 1. "Daredevil" adversary ($\alpha_A = .0, \alpha_P = -1.0$ and $\alpha_T = .1$) chasing the player with a range attack.

weight of the NPC's own empowerment (α_A=.1, α_P=−1.0), the NPC is more inclined to trade-off losses in its own- for the decimation of player empowerment. As a result, we get a *daredevil* adversary[1], chasing and shooting the player as illustrated in Fig. 2. As final scenario, we investigate the adaptivity of our NPC by equipping the player with a range attack action as well. A video[2] shows how the adversary adapts to this new threat by dodging and keeping distance.

This experiment shows that CEM can yield adaptive adversarial behaviour, including the classic predator-and-prey behaviour present in many games. Furthermore, it highlights that the CEM weights should not be considered a burden, but rather a feature to create different personas, thus increasing the believability of our NPCs while overcoming the weaknesses of utility-based agents outlined in the introduction.

D. Experiment II: Exploiting Affordances

In a sufficiently complex game, the wealth of possible interactions between a character's abilities and features of the environment becomes hard to anticipate even for the game's designers. As a consequence, most hand-crafted NPCs do not fully exploit these interactions. In more open-ended algorithms such as MCTS, this anticipation problem creeps into the definition of the optimisation objective, resulting in blunt adversary behaviour. Empowerment is defined on an agent's possible interactions with its world, and should thus be sensitive to *any* interaction between *any* type of "functional content" [14]. In our second experiment, we thus investigate if a CEM-driven NPC can leverage the possible interactions that a game affords to the full extent for adversarial behaviour. Because tiny changes to e.g. the environment or an agent's abilities can turn the emerging gameplay upside down, we start with a simple environment and extend it gradually to investigate CEM-driven adaptation.

Fig. 4a shows the initial state of the environment, where adversary and player face each other in an arena surrounded by lava. If an agent happens to step on the lava, its health decreases by one unit per time step. In order to examine longer interaction sequences, we extend our characters' health to four units ($h_t, h_{max} = 4$). Mediated by *health-performance-consistency*, a decrease in health results in lower empowerment even for small lookaheads. The NPC's 3-step empowerment (Fig. 4b) is thus lower in the lava, and decreases the further away the agent is from the platform, where only few action sequences lead back alive.

Under the default configuration, the NPC closes up to the player and blocks it to reduce the latter's movement and thus empowerment. If we give the NPC the ability to push though, the dynamics change considerably: As illustrated in Fig. 3 and in a video[3], the NPC then destroys the player by pushing it into the lava. Importantly, it blocks the player from returning to the platform, no matter which path the latter chooses. The policy thus captures how the agent's new ability, in interaction with the environment, can be exploited the decrease the player's empowerment – resulting in more challenging and arguably novel gameplay.

But what happens if we give the NPC an action which is typically not associated with adversaries, such as healing? Our next scenario shows that this surprisingly takes the

[1] Video online: youtu.be/MVthwbhUNTA
[2] Video online: youtu.be/9WoMKJAwl6k
[3] Video online: youtu.be/-Stm59llrDs

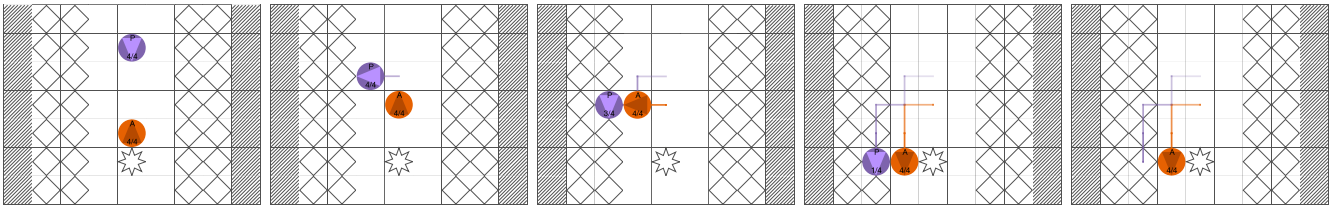
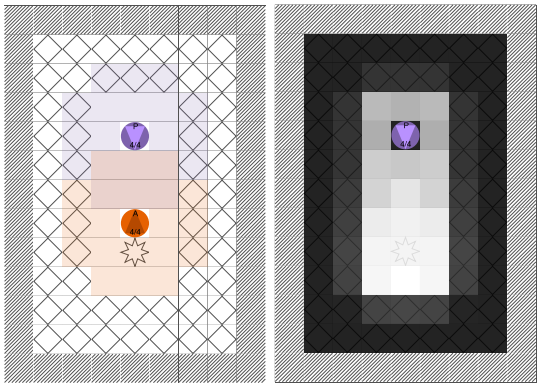


Fig. 3: Experiment 2 (Detail). Adversary pushing player into lava and blocking it from returning to the platform.



(a) Initial state (b) $\mathfrak{E}^{A,3}$

(c) $\mathfrak{E}^{A,3}$, flying (d) $\mathfrak{E}^{A,3}$, recharger

Fig. 4: Experiment 2. Initial state with perceptive field and 3-step empowerment for different modifications.

adverseness of our NPC to a new level: equipped with the ability to heal the player by one health unit per time step, it still pushes the player into the lava. However, once the player is close to ceasing, the NPC uses its healing action to keep the player alive[4]. Crucially, the player's health in this situation would be too low to make it back to the platform. Our CEM-driven NPC thus acts in best super-villain style, and in stark contrast to e.g. MCTS with the only objective to destroy: it just keeps the player's health sufficiently high to exercise control over it – thus optimising its own empowerment – while keeping the player's empowerment low. We can modulate this behaviour by changing the weight parameters: if we reduce α_A, the NPC lets the player die.

Maximising empowerment cannot only be achieved by controlling other characters; in scenarios like the present, it also requires to engage in acts of self-preservation. Dodging attacks by the player as in the previous experiment is such an act. However, previously both agents had identical abilities, which is uncommon in most games. To examine whether CEM can exploit inequalities between characters to further both self-preservation and adverseness, we allow our NPC to range-attack and fly, while the player is limited to melee attacks on the ground. In our testbed, a character that can fly is not affected by the hazardous effect of lava, and the NPC's empowerment is thus not affected by the lava anymore, but only by the surrounding walls and the player (cf. Fig. 4b and 4c). With its new ability, our NPC now dodges the player's melee attacks by escaping over the lava. Once the player veers away from the NPC, it returns and attacks from a distance[5]. Note that, using a uniform model of the player's policy, the NPC expects the player to remain on the platform no more than following it into the lava. However, this would result in a decrease of the player's empowerment – which would be welcome to the NPC trying to minimise it.

In our last scenario, we stress another aspect of self-preservation: not escaping harm, but recovering from it. If we allow our characters to push and perform melee attacks, the NPC engages in close combat, using both direct attacks and pushing the player into the lava. Meanwhile, if we put a recharge unit in the middle of the platform, the characters start competing for the scarce resource: once the NPC's health gets close to zero, it captures the recharge tile to recover, pushing the player off if necessary[6]. The NPC's empowerment heatmap for a lower health state $h_t = 2, h_{max} = 4$ (Fig. 4d) highlights the effect of the recharge station - like a beacon in the reward landscape. This second experiment support our hypothesis that CEM-driven agents can adapt to- and exploit changes in the environment and in their abilities.

E. Experiment III: Distant Threats

The most challenging adversaries are arguably those that strike from a distance, where they remain unaffected by our actions, and potentially also undetected. An NPC that would be inferior in direct combat could cast spells, order air strikes or control traps and doors remotely. In our last experiment, we investigate if CEM allows for such behaviour to emerge.

Key to such behaviour is player and transfer empowerment, with transitions towards direct interactions being facilitated by trading off the NPC's own empowerment. We have designed our last experiment to provoke such a transition and examine the interplay of these three components. In the initial state (Fig. 6a), the player starts on the lower right in

[4]Video online: youtu.be/fy-2hRf-4L8

[5]Video online: youtu.be/tSzYLaCDXiI

[6]Video online: youtu.be/WoWfLRlY2LY

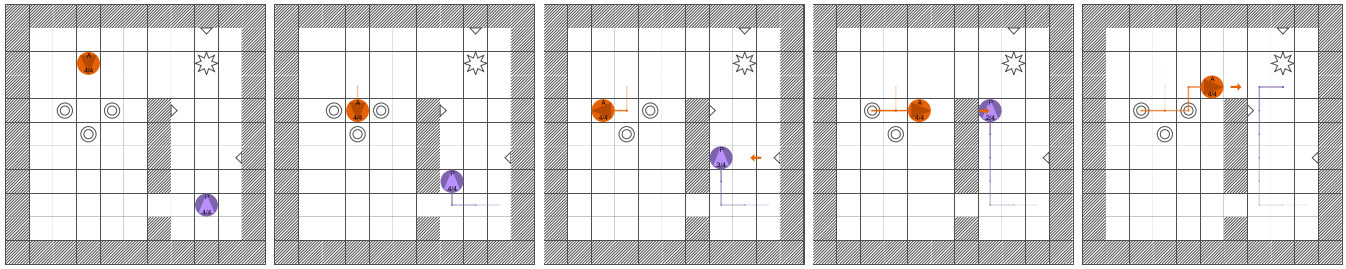
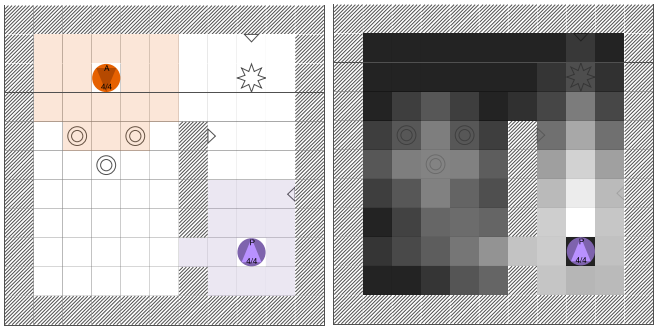
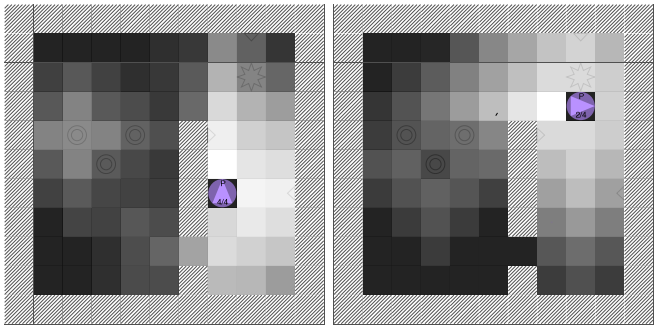


Fig. 5: Experiment 3. Adversary harming player by triggering turrets remotely, eventually destroying it with a range attack.



(a) Initial state (b) $\mathfrak{E}^{T,3}$



(c) $\mathfrak{E}^{T,3}$ (d) $\mathfrak{E}^{T,3}$

Fig. 6: Experiment 3. Initial state, followed by adversary $\mathfrak{E}^{A,n}$ and adversary-player transfer empowerment $\mathfrak{E}^{T,n}$, the latter for three different player positions. Lookahead $n = 3$.

a corridor, while the NPC is situated on the upper left in an open area, separated by a wall with two passages. The player faces three turrets, two on the sides and one ahead. The corresponding triggers are positioned in front of the NPC. Both characters have the ability to perform a range attack.

The NPC's own empowerment in this state does not convey any information about the best trigger to affect the player, as it only quantifies the NPC's influence on its *own* sensor state. Player- and transfer empowerment in contrast both work as proxy to the player's condition: transfer empowerment measures the impact of turret-triggering on the player's health, which is captured in the latter's sensor; the player's health in turn affects the player's empowerment, which can be exploited by the NPC. Figs. 6b – 6d show how transfer empowerment peaks on- and around the triggers for player positions in the shooting range of different turrets.

Following the CEM policy, the NPC triggers the correct turrets to hit the player on its way towards the goal tile (Fig. 5 1–4). When the player moves between turrets, the NPC positions itself where it can strike quickest, i.e. between the triggers. Once the player gets closer to the goal and thus to the open passage towards the adversary, the latter trades off its own- against the increase of transfer empowerment: the spatial proximity of the player results in a transfer empowerment gradient which the NPC could follow to eventually attack the player directly. By doing so however, the NPC risks its own empowerment to be decreased by a counter-attack. In the present configuration, the adversary eventually moves away from the triggers and attacks the player directly (Fig. 5, last and video[7]). Meanwhile, decreasing the NPC's health (h_t=1, h_{max}=2) makes it remain at its current position and shoot the player from a distance[8]. This experiment supports that CEM also yields complex remote interactions.

IV. Discussion

Our CEM-driven NPC not only proved to be very sensitive to changes in the environment and its own abilities; we also demonstrated how small modifications of the weights can switch behavioural patterns, and yield different character types such as opportunists, daredevils and "super-villains".

Our experiments however have also pointed out the importance of incorporating stronger assumptions about the player's policy to yield more believable behaviour. At present, the NPC assumes all player actions in a given state to be equally likely. Thus, while the CEM policy equips the NPC with a drive for survival and self-defense, no such assumption is present in the model of the player's policy. More than that, the adversary-player relationship is one-sided: while the NPC would select its actions to diminish the player's empowerment, the player is not assumed to have a negative bias. We think that empowerment can be used successfully to induce such a bias into the NPC's model of the player's policy, while maintaining the generality of the approach. Importantly, assuming the player to minimise the NPC's empowerment would be short-sighted: unless fighting adversaries contributes explicitly to achieving a game's goal, a human player might be more inclined to evade adversaries than to attack them. Instead, we suggest to go one step further and model the player as maximising empowerment itself. This should yield a good prior particularly in games where progress is aligned with an increase in options and influence. We presently do not represent the player's goal in

[7]Video online: youtu.be/qBTdGCkspA4
[8]Video online: youtu.be/HnVE-IHmGG8

the policy model, but CEM operates implicitly on the player's trajectories towards goal achievement. Ultimately, we expect the quality of adversarial behaviour to increase further when inferring and modelling these goals explicitly.

Another question arising from our experiments is how challenge induced by CEM-driven adversary NPCs can be modulated to produce a well-balanced player experience [15]. Given that our NPCs adapt to changes in their abilities, such modulation can be facilitated by the classic means of balancing the characters' abilities with respect to the environment. But CEM offers additional alternatives. Adjusting the weight parameters allows us to model characters that challenge us in different ways: an adversary can be made aggressive or more cautious, only fighting back if they are confronted directly. Furthermore, noise can be introduced into the NPCs model of the environment dynamics, making it overconfident or insecure about their own and other characters' possible interactions with the world. Finally, biases can also be introduced into the NPC's model of the player's policy, rendering the latter e.g. as anticipated threat or harmless peer. We expect this to yield particularly interesting gameplay in combination with online model learning.

We finally want to address the scalability of CEM. In order to discriminate small effects of the underlying quantities in this study, we have computed coupled empowerment exhaustively. However, this comes with exponential computational complexity, mostly due to the calculation of the forward transitions and the channel capacity. In recent years, several approximations for the maximisation of mutual information, underlying empowerment, have been proposed, drawing on variational inference and deep neural networks [16], [17], [18]. We believe that these are presently the most promising candidates to increase the scalability of CEM. Furthermore, the lookahead in CEM can be increased by utilising macro-actions. In sufficiently large action spaces, Monte-Carlo sampling of action sequences (cf. [19]) is also likely to yield good approximations. Finally, more informed policy models could not only increase the quality of behaviour, but also be used to prune the search tree.

V. Conclusion and Future Work

We have set out to provide an open-ended action policy for NPCs to *leverage any interaction* a game affords, and to *adapt to changes* in a game with the ultimate goal to design more believable characters. In previous work, we have proposed to use CEM to engineer general companion NPCs that yield a large variety of new and potentially surprising, supportive behaviours. In this paper, we have adopted the action policy to give rise to adversarial behaviour. We have shown by means of a qualitative study that minimising the player's empowerment in a CEM policy yields rich adversarial behaviour, based on our NPC's successful exploitation of interaction affordances, and the adaptation to changes to its own- and the player's abilities, as well as to the environment. Our NPC has used its abilities, e.g. to heal, in ways that we would find surprising even in respect to human opponents.

Our study has provided valuable insights towards increasing the believability of our NPCs further, which we plan to use in a quantitative evaluation to conclude this proof-of-concept. We will employ AI playing agents to remove any experimenter bias, and investigate the suitability of CEM to drive both adversarial and supportive behaviour based on objective metrics such as goal achievement, as well as subjective measures of player experience. Given the present observations, we are confident that our participants will not be bored with stereotypical adversary behaviour, but encounter genuinely new and surprising ways to be mean.

Acknowledgments

CG is funded by EPSRC grant [EP/L015846/1] (IGGI). CS is funded by the EU Horizon 2020 programme / Marie Sklodowska-Curie grant 705643. We thank our reviewers for helpful comments.

References

[1] H. Warpefelt, "The Non-Player Character: Exploring the Believability of NPC Presentation and Behavior," Ph.D. dissertation, Stockholm University, 2016.

[2] M. Treanor, A. Zook, M. P. Eladhari, J. Togelius, G. Smith, M. Cook, T. Thompson, B. Magerko, J. Levine, and A. Smith, "AI-Based Game Design Patterns," in *Proc. Conf. FDG*. ACM, 2015.

[3] C. Holmgård, M. C. Green, A. Liapis, and J. Togelius, "Automated Playtesting with Procedural Personas through MCTS with Evolved Heuristics," *IEEE Trans. Games*, 2018.

[4] P.-Y. Oudeyer and F. Kaplan, "How Can We Define Intrinsic Motivation?" in *Proc. Conf. Epigenetic Robotics*, 2008, pp. 93–101.

[5] K. E. Merrick and M. L. Maher, *Motivated Reinforcement Learning. Curious Characters for Multiuser Games*. Springer, 2009.

[6] C. Guckelsberger, C. Salge, and S. Colton, "Intrinsically Motivated General Companion NPCs via Coupled Empowerment Maximisation," in *Proc. Conf. CIG*. IEEE, 2016.

[7] A. S. Klyubin, D. Polani, and C. L. Nehaniv, "Keep Your Options Open: An Information-Based Driving Principle for Sensorimotor Systems," *PLOS One*, vol. 3, no. 12, pp. 1–14, 2008.

[8] C. Salge, C. Glackin, and D. Polani, "Empowerment – an Introduction," *Guided Self-Organization: Inception*, pp. 67–114, 2014.

[9] C. Guckelsberger, C. Salge, R. Saunders, and S. Colton, "Supportive and Antagonistic Behaviour in Distributed Computational Creativity via Coupled Empowerment Maximisation," in *Proc. Conf. ICCC*, 2016, pp. 9–16.

[10] C. Salge and D. Polani, "Empowerment as Replacement for the Three Laws of Robotics," *Frontiers in Robotics and AI*, vol. 4, p. 25, 2017.

[11] R. Blahut, "Computation of Channel Capacity and Rate-Distortion Functions," *IEEE Trans. Information Theory*, vol. 18, no. 4, pp. 460–473, 1972.

[12] S. Arimoto, "An Algorithm for Computing the Capacity of Arbitrary Discrete Memoryless Channels," *IEEE Trans. Information Theory*, vol. 18, no. 1, pp. 14–20, 1972.

[13] T. M. Cover and J. A. Thomas, *Elements of Information Theory*, 2nd ed. Wiley-Interscience, 2006.

[14] G. Smith, "Understanding Procedural Content Generation: A Design-Centric Analysis of the Role of PCG in Games," in *Proc. Conf. CHI*. ACM, 2014, pp. 917–926.

[15] A. Denisova, C. Guckelsberger, and D. Zendle, "Challenge In Digital Games: Towards Developing a Measurement Tool," in *Proc. Conf. CHI*. ACM, 2017, pp. 2511–2519.

[16] S. Mohamed and D. J. Rezende, "Variational Information Maximisation for Intrinsically Motivated Reinforcement Learning," in *Proc. Conf. NIPS*, 2015, pp. 2125–2133.

[17] K. Gregor, D. J. Rezende, and D. Wierstra, "Variational Intrinsic Control," *arXiv preprint arXiv:1611.07507*, 2016.

[18] M. Karl, M. Soelch, P. Becker-Ehmck, D. Benbouzid, P. van der Smagt, and J. Bayer, "Unsupervised Real-Time Control through Variational Empowerment," *arXiv preprint arXiv:1710.05101*, 2017.

[19] C. Salge, C. Glackin, and D. Polani, "Changing the Environment Based on Empowerment as Intrinsic Motivation," *Entropy*, vol. 16, no. 5, pp. 2789–2819, 2014.

Accelerating Empowerment Computation with UCT Tree Search

Christoph Salge
School of Computer Science
University of Hertfordshire
Hatfield, UK
c.salge@herts.ac.uk

Christian Guckelsberger
Computational Creativity Group
Goldsmiths, University of London
London, UK
c.guckelsberger@gold.ac.uk

Rodrigo Canaan
Tandon School of Engineering
New York University
Brooklyn, NY, USA
rodrigo.canaan@nyu.edu

Tobias Mahlmann
t.mahlmann@gmail.com

***Abstract*—Models of intrinsic motivation present an important means to produce sensible behaviour in the absence of extrinsic rewards. Applications in video games are varied, and range from intrinsically motivated general game-playing agents to non-player characters such as companions and enemies. The information-theoretic quantity of *Empowerment* is a particularly promising candidate motivation to produce believable, generic and robust behaviour. However, while it can be used in the absence of external reward functions that would need to be crafted and learned, empowerment is computationally expensive. In this paper, we propose a modified UCT tree search method to mitigate empowerment's computational complexity in discrete and deterministic scenarios. We demonstrate how to modify a *Monte-Carlo Search Tree with UCT* to realise empowerment maximisation, and discuss three additional modifications that facilitate better sampling. We evaluate the approach both quantitatively, by analysing how close our approach gets to the baseline of exhaustive empowerment computation with varying amounts of computational resources, and qualitatively, by analysing the resulting behaviour in a *Minecraft*-like scenario.**

***Index Terms*—empowerment, tree search, MCTS, UCT, Minecraft, intrinsic motivation**

I. Introduction

The empowerment formalism [1], [2] offers interesting game applications in terms of believable NPC behaviour [3], general game-play [4] and player experience modelling [5]. But the high computational complexity of empowerment is problematic for a wider application in games. In this paper we show how a UCT tree search formalism [6], [7] can be adapted to approximate empowerment maximisation in the discrete domain. But first, we motivate the application of empowerment to games in more detail.

A. Motivation

Empowerment is a measure of how much an agent can affect the world it itself perceives. Empowerment maximisation is considered an intrinsic motivation (IM) [8], and has been recently linked to competence and autonomy, two motivations which are frequently discussed in a games context [9]. As intrinsic motivation, i.e. as an essential motivation linked to agency itself, empowerment can generate behaviour even in the absence of externally defined goals. Behaviour then results from fulfilling a motivation that arises from the agent-world interaction. An illustrative, non-empowerment example for this is the work of Merrick and Maher [10], [11], where an agent's actions are selected based on learning progress [12] and curiosity [13]. Curiosity, or the desire to experience something new, can create behaviour without further reward. The broad concept of curiosity is also a good illustration of an *intrinsic* motivation, as it is hard to imagine agency without the least desire to experience novelty or learn something new. Empowerment, in contrast, is about having affordances, about self-efficacy and the ability to affect one's own world. Empowerment has also been linked to the idea of an organism's striving to preserve its precarious existence [14]. Applied to a Minecraft-like simulation this drive to self-preservation resulted in behaviour where the agent would restructure the world to keep itself alive [15], producing different behaviour patterns in reaction to changes in the environment. The same work also demonstrated how the embodiment of the agent was reflected in the structures built in the world. This apparent self-directed behaviour, arising from- and reacting to changes in the game world make the application of empowerment in games interesting.

Previously, empowerment has been applied to play Sokoban and PacMan [4]. In this work, Anthony *et al.* speak about the generality of empowerment by asserting that it provides a utility that: "1) derives only from the structure of the problem itself and not from an external reward; 2) identifies the desirability of states in a way that matches intuition; and 3) carries over between scenarios of apparently different character." This would make empowerment a good proxy for general game-play, and thus biasing the decision making of reward-optimising agents with empowerment might lead to better performance. Similar approaches have been used in the domain of robotics, where a robotic follower [16] and underwater vehicles [17] had their decision making biased or enhanced with empowerment maximisation. But while the empowerment formalism is generally applicable, i.e. can be computed based just on the structure of a given forward model, the utility it provides may not always agree with an externally defined reward. One common example here are games where empowerment has to be "traded away" to win a game - imagine a game like Starcraft where you first obtain resources and units, which increase your empowerment, but then you lose those units and resources in a material exchange to defeat

978-1-5386-4359-4/18/$31.00 ©2018 IEEE

your opponent. Note though, that there is still an incentive to maximise empowerment, even as you are giving it up. It is also possible to define games with a (possibly contrived) win condition that conflicts with empowerment. In a lot of cases, though, games are designed to be aligned with intrinsic motivations, and progressing in a game usually goes along with increasing player empowerment. As you progress you usually get more abilities, better and more tools, have access to more actions, can explore more of the world and generally have more rather than less influence on the game world.

Creating believable non-player characters (NPCs) in a game is another possible application of empowerment in games. Empowerment maximisation can, without defined goals, produce behaviour related to self-preservation and maximisation of options. This can be used to give NPCs an appearance of self-determination. The general applicability of the formalism also allows NPCs to adapt to changing circumstances. This has been explored by Guckelsberger *et.al.* [3] for the design of general companion characters. In addition to having the companion NPC maximise their own empowerment, two additional empowerment drives were introduced. Maximising the player's empowerment motivates the companion to protect and help. It would, for example, shoot enemies that threaten the player. This multi-perspective approach has also been explored in relation to robots, where it could produce generic robot behaviour guidelines [18]. Very recently, it has been extended to design highly adaptive and robust adversary NPCs [19].

Finally, there is also the question as to which extent a player's empowerment in a game can be used as predictor for their experience. A preliminary study [5] identified causal efficacy as potential candidate experience that empowerment is closely related to, with mediate effects on "challenge, involvement, attention and engagement, learning and emotions". Having a measure that computes user experience without an actual player would be beneficial in rapid prototyping and when creating or adapting games automatically.

Yet, one downside of empowerment maximisation is its lack of scalability, due to the formalism's high computational complexity, especially when looking at longer time horizons. Approximations have been developed both for the *continuous* domain [20] and discrete but *noisy* models [4]. In this paper, we use UCT to accelerate the computation of the most empowered action in a *discrete and deterministic* model.

B. Overview

We first describe the actual empowerment formalism, and then focus on empowerment in discrete and deterministic models. We discuss the problems arising from sparse sampling, and introduce a modified UCT tree search algorithm to find the most empowered actions with less sampling. We complement this with three modifications – *novelty bias*, *aggregated empowerment* and *full branching* – to further enhance the sampling. We evaluate the optimisation scheme in a Minecraft-like world model which has also been used in previous work [15]. We briefly introduce the model, followed by a quantitative and a qualitative evaluation. We demonstrate how the UCT approach and the different modifications perform better with less samples than the current baseline of sparse random sampling.

II. Empowerment

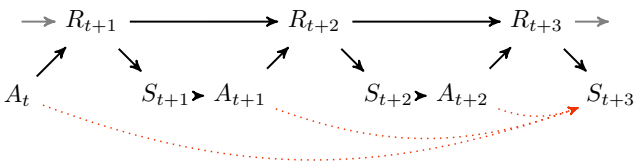


Figure 1. The perception-action-loop visualised as a Bayesian network. *S* is the sensor, *A* is the actuator, and *R* represents the rest of the system. The index t indicates the time at which the variable is considered. This model is a minimal model for a simple memoryless agent. The red arrows indicate the direction of the potential causal flow relevant for 3-step empowerment.

Empowerment [1], [15] is an information-theoretic formalism that captures how much an agent can affect the world it itself perceives. It is defined for all systems that can be modelled as an action-perception loop, as seen in Fig. 1. Where the random variables S, A and R model the sensors, actions and remaining state of the world, respectively. Empowerment for a given state $r \in R$ is formally defined as the channel capacity from an agent's actions at time t to its sensors at a later point in time. This channel goes through the environment R. A common generalisation is n-step empowerment, where all actions from a_t to a_{t+n-1} are considered as input to the channel, and the output is the sensor of the agent at $t+n$:

$$\mathfrak{E}(r_t) = \max_{p(a_{t\ldots t+n-1})} I(S_{t+n}; A_{t\ldots t+n-1}|r_t). \qquad (1)$$

The quantity captures how much information an agent can "inject" into its sensor S_{t+n} via the environment by intervening earlier in $A_{t\ldots t+n-1}$. It is equivalent to potential causal information flow as defined in [21]. An agent is usually highly empowered if it has a lot of different options that all lead to different, predictable outcomes, unaffected by noise. A highly empowered agent can reliably bring about many different sensor states. A more detailed discussion of the general concept and its information theoretic basis can be found in [1], [15].

Empowerment maximisation is the idea that an agent wants to be in a state that is highly empowered. Note, when computing the empowerment for a given state the channel capacity achieving distribution $p(a_{t\ldots t+n-1})$ might contain a lot of action sequences that lead to bad outcomes. But the action policy, i.e the way the agent picks it actions, is not determined by this distribution. Instead, a greedy empowerment maximisation strategy computes the empowerment for all possible successor states to the current states, and then chooses the action leading to the one with the most empowerment. It is empowerment maximisation that is considered an intrinsic motivation, and

in this paper, we focus on how to efficiently determine which actions will lead us to the most empowered state.

While empowerment can be defined for both a noisy and even a continuous channel, in this paper we focus on discrete and deterministic models. In the deterministic case, where each possible action sequence $a_t, ..., a_{t+n-1}$ leads to one specific state s_{t+n}, the channel capacity is the logarithm of the number all reachable states. So, to calculate the empowerment we have to determine the resulting sensor state for each possible n-step action sequence, and then count how many different states there are in total. This simplifies the computation significantly, and allows to compute n-step empowerment for larger time horizons n. We will refer to these reachable sensor states as "reachable states", dropping the word sensor for brevity. Despite this simplification, if we look at a model where each action step has, for example, a branching factor of 5, we still have to evaluate 5^n action sequences and the corresponding final states. This number quickly grows infeasibly. In previous work [15] this was addressed with sub-sampling, where a random subset of all possible action sequences was evaluated to compute 15-step empowerment, with a branching factor of 12. In this work the limitations of random sampling became evident. Sometimes, the empowerment-maximising agent would be in a situation where it could get to a part of the world where it would be able to reach a lot of different sensor states, but to get there it would have to perform very specific actions in the beginning of the action sequence - think of a bridge as an evocative example. Due to the random nature of the sampling, this bridge might only be crossed with a few of the sequences, and the evaluation would miss the considerable gain that going over the bridge yields for the agent's empowerment. In this work, we aim to use UCT tree search to both (i) bias the exploration towards those initial sequences, and to (ii) identify the best possible action more efficiently.

III. Empowerment with UCT Tree Search

In this section we outline how to use tree search with UCT (upper confidence bound applied to tree search, [6]) to accelerate the decision making based on deterministic empowerment. To find the best action, we need to determine which of the successor states of the current world state has the most different sensor states reachable with action sequences of length n. Note that while we compute the empowerment for sensor states derived from the world, the computations to determine the empowerment are done with a complete model of all world state transitions. In other words, our computation is not limited by the agent's perspective and has access to the full world model.

Our approach is inspired by Monte Carlo tree search (MCTS) with UCT [7], but there are substantial adaptations in the expansion, simulation and backpropagation steps. The basic idea of MCTS UCT, or any informed search for that matter, is to guide the use of resources, such as forward model calls, to the parts of the search space which yield the most information for picking the best action.

So, when we sample action sequences to determine reachable sensor states, we assume that sequences starting with actions that have led to new results are more likely to yield new results again. We then use the UCT formula [6] to bias our exploration towards those actions that have previously led to new states. Further analysis is still needed to determine if the mathematical properties of the bandit problem, which UCT is derived from, hold for empowerment computation. Here we look at simulated results only.

In the next section, we describe the algorithm in detail, and motivate three modifications. Both are further illustrated with pseudocode in Alg. 1. Keep in mind, the goals is to determine which successor state of the current world state has the most reachable sensor states, i.e. is the most empowered.

A. UCT tree search

In our algorithm nodes are associated with world states. We start by creating a root node that is associated with the world in its current state. This node has a depth of zero.

1) Expansion: The algorithm starts at the root, and checks if there are unexpanded, i.e. unvisited children. As long as there are unvisited children we select randomly one of the actions that would lead to an unexpanded child. We create the child node with a depth value one higher than its parent node. We then repeat the expansion step, i.e. expanding another unvisited child, until we reach a node that has a depth equal to the empowerment horizon n plus one. This is because the first step just creates the successor nodes that we are evaluating for their empowerment, and the successive n steps realise the n-step empowerment approximation. Note that we immediately expand the tree down to n-steps, and there is no roll-out phase.

2) Backpropagation: Once we reach a node at the horizon, we obtain the agent's sensor state and store it in the set of reachable states in the node. We then also add this reachable state to the node's parent node. The parent checks if it already has this state in its reachability set, and if not, adds it. It then also adds it to its own parent, recursively. After the backpropagation finishes, the tree should be in a state where each node has a reachability set that contains all sensor states that can be reached from it with the already expanded action sequences. After this step the algorithm starts over at the root of the tree, if there is time left.

3) Selection: As the tree fills up, the algorithm will encounter nodes where all children have at least been visited once, and it then has to decide which node to expand again. At this point we sort the children c with the modified UCT formula and pick the child c with the highest value:

$$uct(c) = \frac{c.states.size()}{c.visits} + 0.01 \cdot \sqrt{\frac{\log(root.visits)}{c.visits}} \quad (2)$$

The function $states.size()$ gives us the size of the reachability set, and $visits()$ tells us how often the root node and the child node have been visited. The size of the reachability set divided by the visits to the child gives us a value between 0.0 and 1.0, the ratio of how many new states each visit to this

Algorithm 1 Overview of the agent's decision making algorithm. Methods that take place in other objects, such as applying actions to the world or selecting a random number are omitted for brevity. HORIZON is $n+1$ for n-step empowerment, as we can determine the initial successor states in the same tree. The colours indicate code changes for the modification. For basic UCT empowerment read just the code in black. For *aggregated empowerment*, add the red line at 18. For *novelty bias* add the two blue code snippets in 29 and 36. For the *full branching* modification add the code in green, and set DEPTH to a non-zero value.

```
1:  procedure BEST_ACTION(World w)
2:      Node root ← new Node
3:      while time left do
4:          depth ← 0
5:          Node t ← root
6:          World test ←copy(w)
7:          while depth < HORIZON - DEPTH do
8:              t.visits++
9:              depth++
10:             if t has unexpanded children then
11:                 Action a ← RANDOM_ACTION(t, test)
12:                 test.applyAction(a)
13:                 Node child ← new Node
14:                 child.parent ← t
15:                 child.action ← a
16:                 t.children.add(child)
17:                 state ← test.s()          ▷ Agent sensor state
18:                 ADD_STATE(state, child)
19:                 t ← child
20:             else                          ▷ use UCT selection
21:                 t ← BEST_CHILD(t, root)
22:                 test.applyAction(t.action)
23:         BRANCH(t, test, DEPTH, root)
24:     return root.children[max(states)].action
25:
26: procedure ADD_STATE(state, Node n)
27:     if state ∉ n.states then
28:         n.states.add(state)
29:         if state ∉ parent.states then n.unique++
30:         if ∃ n.parent then ADD_STATE(state, n.parent)
31:
32: procedure BEST_CHILD(Node t, Node root)
33:     best ← null
34:     fitness ← 0
35:     for c ∈ t.children do
36:         f = (|c.states| + c.unique)/c.visits + 0.01 · sqrt(log root.visits / c.visits)
37:         if f > fitness then fitness ← f; best ← c
38:     return best
39:
40: procedure BRANCH(Node t, World w, d, Node root)
41:     if d = 0 then
42:         state ← test.s()                  ▷ Agent sensor state
43:         ADD_STATE(state, t, root)
44:     else
45:         Action[] a ← w.getPossibleActions()
46:         for action ∈ a do
47:             World test ← copy(w)
48:             test.applyAction(action)
49:             Node child ← new Node
50:             child.parent ← t
51:             child.action ← action
52:             t.children.add(child)
53:             BRANCH(child, test, d - 1, root)
54:
55: procedure RANDOM_ACTION(Node t, World w)
56:     Action[] a ← w.getPossibleActions()
57:     for actions ∈ t.children do a.remove(action)
58:     return random(a)
```

particular child found. The best value here is 1.0, meaning every visit leads to one new sensor state. The term in the square root guides selection towards under-explored states; it grows larger as more action sequences are sampled that do not go through c. The factor of 0.01 was chosen to have a good balance where the distribution of visits among children was neither approximately uniform (as it would be for a larger value), nor heavily skewed towards the first best solution (as it would be for a smaller value).

4) Action Selection: At some point the algorithm runs out of time (or some other computational limiter) and it needs to decide what action to take. It looks at the root and picks the action leading to the child with the biggest reachability set. This action should lead to a state with the highest n-step empowerment, meaning that from this state, the agent can reach the maximum number of different sensor states within n steps. Note that the algorithm does not select the state with the highest ratio of new states vs. visits, which was used for the UCT based selection.

Rationale: Compared with randomly sampling action sequences for each successor state, this algorithm should perform better at finding the state with the highest empowerment, as it spends less time on investigating the worst alternatives. The UCT function in selection should bias computational resources towards the most promising candidates. While the utility of a node for action selection is the size of the reachability set, we chose to divide this value by the number of visits for the selection phase. We thus keep the value between 0.0 and 1.0 and the nodes remain comparable. If this was not the case, nodes that had been explored more would be preferred as the number of found reachable states heavily depends on the number of visits.

Before we evaluate this approach, referred to as UCT from now on, we introduce three modifications to the algorithm.

B. UCT with Novelty Bias

For this modification we check every time a child node adds a reachable state to its parents reachability set, if this state is new to the set. If this is the case, no other child of that parent has an expanded action sequence leading to that sensor state

yet. In case of a novel addition, the child increases its novelty counter by one (line 29 in Alg. 1).

The novelty counter is added to the size of the reachability set in the UCT function. This means that the performance of the nodes is now not only defined by how many different states it found, but also by how many of those states were novel contribution to the reachability set of its parent node; an idea somewhat similar to novelty pruning[22].

Rationale: This modification is aimed at biasing exploration towards sub-sequences that add novel states. Consider an agent in front of a doorway. One action leads through it, while the others keep the agent moving around in the room. The sequences that stay in the room end up overlapping, and while they might each lead to a sizeable amount of states, they all lead to the same states. In contrast, sequences that go through the door can add new reachable states to the parent nodes, and should therefore be preferred.

C. Aggregated Empowerment

For aggregated empowerment, which we will refer to as UCTa, we not only consider the sensor states reached at tree depth n, but we also consider all reachable states along the way. This is achieved by simply adding the sensor state of the current world state to every newly expanded node and propagating it upward (line 18 in Alg.1).

Rationale: Aggregated empowerment corresponds to a somewhat different quantity, and the implications of this measure go beyond the scope of this paper. Using this value to approximate regular empowerment however still has the advantages that it allows to differentiate between individual action sequences. Normally, each sequence reaches exactly one state. But with aggregated empowerment, sequences that go through different sensor sequences along the way are considered better. This difference can indicate that the agent is still operational and able to affect the world.

More conceptually, it also allows to differentiate between different sequences, even if they ultimately end in the same state, e.g. death. In this case, the agents prefers to live a less boring life. In a way, this introduces a form of count-based novelty into the empowerment calculation.

D. UCT with Full Branching

For n-step empowerment with k-step *full branching*, the algorithm only expands the tree to a depth of $n - k$, i.e. it stops k steps before it reaches the time horizon. It then fully expands the tree from that node for the remaining k steps with depth-first search, and eventually propagates all found sensor states upwards. All full branching examples in this paper use 1-step full branching. This modification is implemented by the extension of the **BRANCH** function in Alg.1, highlighted in green.

Rationale: Full branching also aims to differentiate between sequences. It basically computes 1-step deterministic empowerment for the semi-leaf node. This local empowerment ideally gives us an idea of how empowered close-by states are, and thus should guide our exploration towards- or away from action sequences starting with the same actions.

IV. Evaluation

A. Simulation Model

To evaluate the different UCT algorithms we implemented a three-dimensional block world, similar to [15]. The world is a three dimensional grid, and each grid cell is empty or contains exactly one block such as earth, the agent and lava. The agent can try to move in the four cardinal directions (*north, east, west, south*). The move will be successful, if a) the target location is empty, or if b) the target location is filled, but the one above is empty. In the second case, the agent will 'climb' to the location above the target block. In all other cases the move fails (is blocked) and the agent remains in its current position. The agent can also *act* in all six directions (*above, below, north, south, east, west*). This action is context-sensitive on the agent's inventory, which can hold exactly one block. If the inventory is empty, the agent will try to take the block from the target location into its inventory, if there is any. If the inventory contains a block, the agent will try to place it. This will succeed if the target location is empty. If an action fails, the world remains unchanged. There are two additional actions, *waiting* and *destroying the current block* in the inventory. Overall, this gives the agent 12 actions in each time step.

Between actions the world simulates liquid flow and gravity. Agents and lava are affected by gravity, i.e. they fall until they rest above a filled block. Earth blocks are not affected by gravity. Lava spreads to neighbouring tiles every 4 time steps, if they are empty. Lava is an environmental hazard, and the agent dies if it is next to a lava block. 'Death' in this case means that the agent's actions have no effect any more on the world. The sensors considered for the empowerment computation capture the agent's x, y, z position.

B. Method

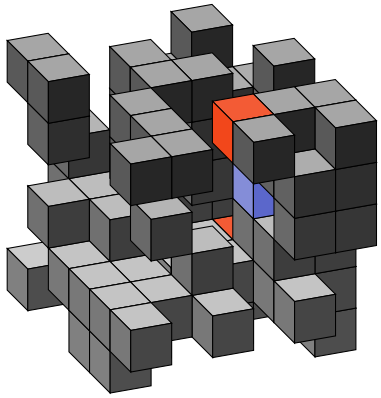


Fig. 2: Typical randomly generated world used for the quantitative evaluation. Two red lava blocks are visible among the grey earth blocks. The agent is colored blue.

For the quantitative evaluation we created 1000 different random worlds of size 7x7x7. Each block has a 40% chance to be an earth block, a 2% chance to be a lava block, and remains empty otherwise. Fig. 2 shows an example world.

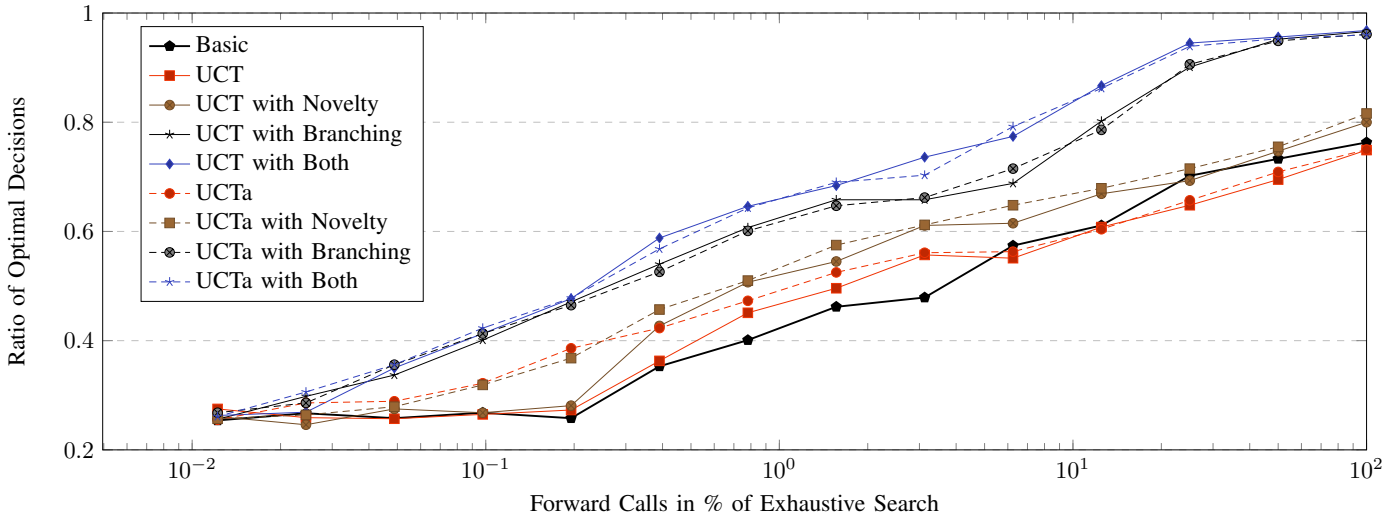


Fig. 3: Ratio of selecting the optimal action for different budgets of forward model calls. The evaluation is based on 1000 different random worlds, computing 4-step action sequences.

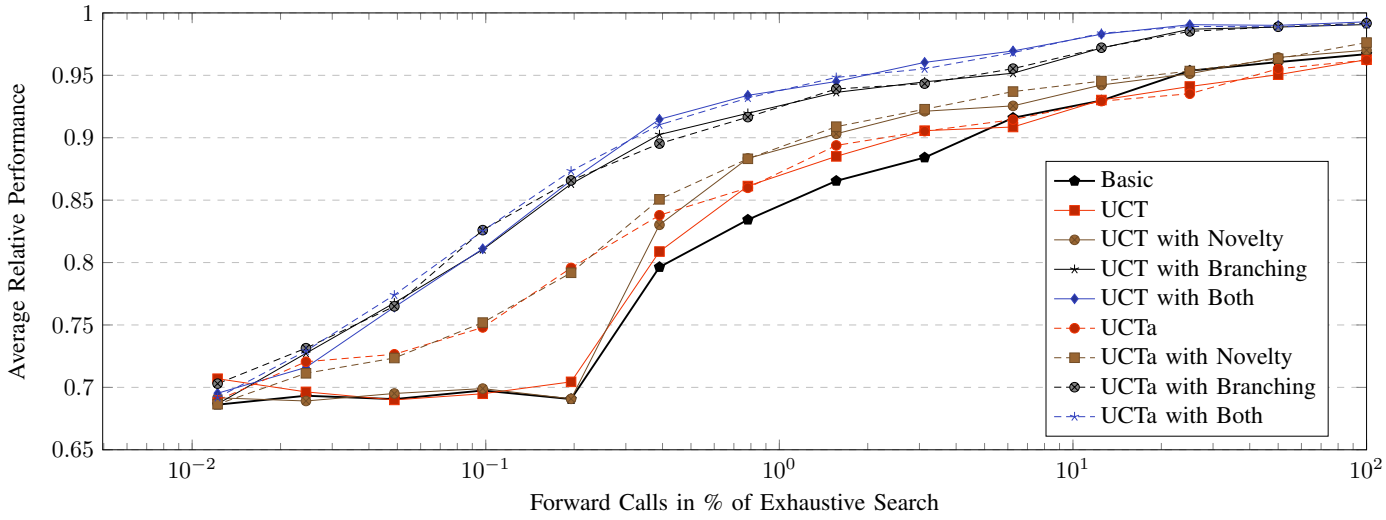


Fig. 4: Average relative performance (in reachable states) for the picked actions compared to the optimal action, for different budgets of forward model calls. The evaluation is based on 1000 different random worlds, computing 4-step action sequences.

The agent (blue) is placed in a random position, replacing the block in that position if necessary.

For each world we computed 4-step empowerment with an exhaustive depth-first search to obtain a baseline comparison. We recorded the number of reachable states, which determine the empowerment, for each immediate successor state and the action leading to that state from the root. This gives us an empowerment value for each action. The UCT algorithm can ultimately be applied to longer action sequences, but we chose a horizon of 4 for the quantitative evaluation, as it allows us to compare the approximations to an exact baseline.

We evaluated 9 different algorithms. The *basic* agent realises just random sampling, as describe in [15]. We evaluated the described *UCT* algorithm with each of the three modifications (*novelty bias*, *aggregated empowerment*, *full branching* for 1 step) turned on or off. Additionally, we evaluated the *UCTa* algorithm based on aggregated empowerment for each modification.

For each of the 1000 different worlds, each of the 9 algorithms tried to find the most empowered action. Our goal was to see how the algorithm's performance would degrade if they were given less computational resources. To have a hardware-independent comparison, we limited each algorithm to a certain number of forward model calls, which are used every time a world is advanced by applying an action. This decision was based on a preliminary analysis, showing that forward model calls were (unsurprisingly) the main contributor to computational load. We recorded the number of forward model calls used by the depth-first algorithm from the exhaustive baseline computation. This number, 22621, was considered as

100% of needed calls. We then evaluated all 9 algorithms on all 1000 worlds by giving them only $1/2$, $1/4$, $1/8$, etc, of the forward calls required for the baseline.

The performance of returned actions was evaluated in two ways. We checked, in comparison to the baseline computation, if the action selected by an action is optimal, i.e. if it has as many reachable states as the best answer. The average performance in this case is the ratio of how many of the chosen actions where optimal. We also implemented a second performance measure, where we compared the empowerment of the given action to the empowerment of the best action. If for example the best action would lead to a state with 20 reachable states, and the given answer would lead to a state with 10 reachable states, the performance would be 0.5. The performance is then averaged over the 1000 tested worlds.

C. Results

Figs. 3 and 4 show the results of the evaluation. As expected, both performance measures get worse if the agents get fewer forward calls to determine the best action. The important result here is that the UCT algorithms, especially those with modifications, outperform the basic agent from [15] for less forward calls. If we look at the graph in more detail, we see that for the least number of samples, all agents perform at about 0.7 in Fig. 4 and at about 0.25 in Fig. 3. This is basically the performance of a random agent that occasionally finds the best action by chance, or picks an action that has decent empowerment. Keep in mind that in some worlds several actions are optimal, or close to optimal.

The remaining graph can be split into three parts. For the first 5 sample points, i.e. the part with the least forward calls, we see that the basic agent —●—, as well as UCT —■— and UCT with novelty —●— all perform at the random level. At this point so few forward calls are made that each child of the root is only expanded once, i.e. has one full actions sequence going through it evaluated. This does not allow yet for an informed choice, as all sequences in this case lead to exactly one sensor state, making all choices equivalent. The algorithms with aggregated empowerment (-●-,-■-,-⊗-,-✶-) perform better here, as even a single sequence becomes informative. Each sequence might be visiting more or fewer different states, giving a better picture of how much an agent that picked a specific first action can affect the world afterwards. If, for example, the first action would lead the agent to touch lava and die, or fall in a hole, then all successive states of that sequence would be identical. The four algorithms with full 1-step branching (—✶—,—♦—,-⊗-,-✶-) perform best in the segment, as they create the most information for a single sequence. By evaluating how many states could be reached from the second to last step, the algorithms can differentiate between different starting actions. These approaches cost slightly more, as they use 11 additional forward calls in the end, but then saves forward calls by not having to go down the same full sequence several times.

In the middle segment of the graph we see an increase in performance for all algorithms. Noteworthy here is that for every configuration the variant with novelty outperforms the respective variant without novelty bias. As we now have enough samples so that children are fully expanded, the UCT selection comes into play. The bias towards those states that brought novel contributions seems to guide us towards better states. Aggregated Empowerment seems to have little effect on performance in this segment.

Towards the end, where the algorithms get nearly enough calls to exhaustively sample the space we can see that the marginal difference in performance gets slimmer as the performances increase overall. The algorithms basically split into two groups, those with full 1-step branching (—✶—,—♦—,-⊗-,-✶-) clearly outperforming those without. We should also note that the basic agent performing random sampling gets more competitive once we get to nearly 100% of the needed samples.

D. Bridge Example

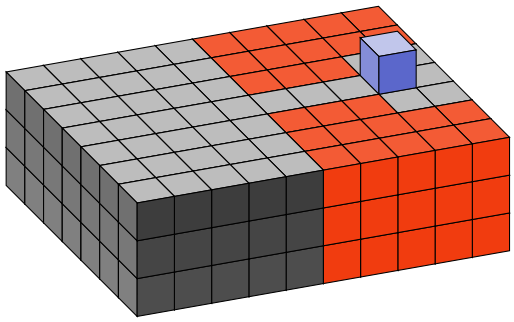


Fig. 5: Bridge example world. The agent (blue) should cross over the narrow bridge and avoid the lava (red).

The qualitative effect of the different algorithms becomes more evident if we are looking at a concrete example. In Fig. 5 we see an agent on a small platform surrounded by lava. The agent should cross over the narrow bridge to reach the much larger area. This would massively increase its empowerment. Looking at 10-step empowerment, the agent has a long enough time horizon to figure this out. But it is infeasible to evaluate all 61,917,364,224 10-step action sequences, so we do not have a ground truth to evaluate against. This is the kind of scenario the algorithm in this paper was developed for.

We evaluated this example with 10000 forward calls and less, for all previously described algorithmic variations. We ran the world seen in Fig. 5 100 times, and we checked how often the agent started moving onto the bridge leading to more empowerment. From our conceptual understanding of discrete empowerment we know that this is the most empowered option. The graph in Fig. 6 shows the results of the evaluation. The basic —●— and the UCT agent —■— struggle to find the optimal action, as fewer than 5% actually move towards the bridge with the first action. The bridge defines a bottleneck that first has to be crossed with an initial, very specific 4-step sequence, and biasing the exploration of the graph towards this one sequence does not happen in this case. The addition of novelty bias seems to add relatively little, but full branching

and aggregated empowerment pick the optimal action far more often. The algorithm that combines all modifications performs best, finding the optimal solution 52% of the time compared to 2% for the basic agent from [15]. As a side note, this is also an example of how directed behaviour arises in a world without the kind of utility function typical to most games. The agent prefers to be on the bigger platform, as it can move around more and dig down to obtain blocks to climb up.

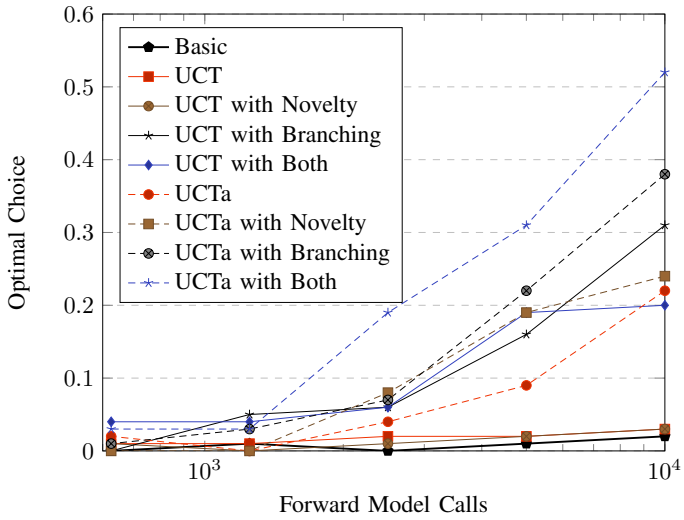


Fig. 6: Evaluation of the bridge example in Fig. 5 for 10-step sequences. The optimal choice is going towards the bridge.

V. Conclusion

The results indicate that all three modifications introduced in this paper improve the performance of empowerment maximising agents that have a limited amount of computational resources. While the approach is intended to be used for longer, i.e. above 10-steps, action sequences, we evaluated 4-step sequences. Due to computational intractability, we do not have a baseline comparison for longer sequences. The bridge example is promising though, as it showed that the modified algorithms can deal with bottlenecks in longer sequences, something the basic algorithm from previous work [15] struggled with considerably.

A. Future Work

This technical improvement opens up possible applications of empowerment in the discrete and deterministic domain. For example, this would make it more feasible to apply an empowerment-biased agent to games such as those found in general game-playing competitions [23], [24]. The other possible extension of this work is to extend it to non-deterministic but discrete models. While there are faster approximations for this domain [4], it might prove useful to apply the extensive catalogue of MCTS enhancements to this problem.

Acknowledgement

CS is funded by the EU Horizon 2020 programme under the Marie Sklodowska-Curie grant 705643. CG is funded by EPSRC grant [EP/L015846/1] (IGGI). RC gratefully acknowledges the financial support from Honda Research Institute Europe (HRI-EU). We thank the anonymous reviewers for helpful feedback.

References

[1] A. S. Klyubin, D. Polani, and C. L. Nehaniv, "Keep Your Options Open: An Information-Based Driving Principle for Sensorimotor Systems," *PloS one*, vol. 3, no. 12, pp. 1–14, 2008.

[2] C. Salge, C. Glackin, and D. Polani, "Empowerment – an Introduction," *Guided Self-Organization: Inception*, pp. 67–114, 2014.

[3] C. Guckelsberger, C. Salge, and S. Colton, "Intrinsically Motivated General Companion NPCs via Coupled Empowerment Maximisation," in *Proc. Conf. CIG*. IEEE, 2016, pp. 1–8.

[4] T. Anthony, D. Polani, and C. L. Nehaniv, "General Self-Motivation and Strategy Identification: Case Studies based on Sokoban and Pac-Man," *Trans. CIAIG*, vol. 6, no. 1, pp. 1–17, 2014.

[5] C. Guckelsberger, C. Salge, J. Gow, and P. Cairns, "Predicting Player Experience Without the Player.: An Exploratory Study," in *Proc. Symp. Computer-Human Interaction in Play*. ACM, 2017, pp. 305–315.

[6] L. Kocsis and C. Szepesvári, "Bandit Based Monte-Carlo Planning," in *Machine Learning: ECML 2006*. Springer, 2006, pp. 282–293.

[7] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A Survey of Monte Carlo Tree Search Methods," *Trans. CIAIG*, vol. 4, no. 1, pp. 1–43, 2012.

[8] P.-Y. Oudeyer and F. Kaplan, "How Can We Define Intrinsic Motivation?" in *Proc. 8th Conf. Epigenetic Robotics*, 2008, pp. 93–101.

[9] S. Roohi, J. Takatalo, C. Guckelsberger, and P. Hämäläinen, "Review of Intrinsic Motivation in Simulation-based Game Testing," in *Proc. 36th ACM Conf. Human Factors in Computing Systems*, 2018.

[10] K. Merrick, "Modeling Motivation for Adaptive Nonplayer Characters in Dynamic Computer Game Worlds," *CiE*, vol. 5, no. 4, p. 1, 2008.

[11] K. E. Merrick and M. L. Maher, *Motivated Reinforcement Learning. Curious Characters for Multiuser Games*. Springer, 2009.

[12] P. Y. Oudeyer, F. Kaplan, and V. V. Hafner, "Intrinsic Motivation Systems for Autonomous Mental Development," *IEEE Trans. Evolutionary Computation*, vol. 11, no. 2, pp. 265–286, April 2007.

[13] J. Schmidhuber, "Developmental Robotics, Optimal Artificial Curiosity, Creativity, Music, and the Fine Arts," *Connection Science*, vol. 18, no. 2, pp. 173–187, 2006.

[14] C. Guckelsberger and C. Salge, "Does Empowerment Maximisation Allow for Enactive Artificial Agents?" in *Proc. Conf. ALIFE*, 2016.

[15] C. Salge, C. Glackin, and D. Polani, "Changing the Environment Based on Empowerment as Intrinsic Motivation," *Entropy*, vol. 16, no. 5, pp. 2789–2819, 2014.

[16] A. Leu, D. Ristić-Durrant, S. Slavnić, C. Glackin, C. Salge, D. Polani, A. Badii, A. Khan, and R. Raval, "CORBYS Cognitive Control Architecture for Robotic Follower," in *Proc. Symp. System Integration*. IEEE/SICE, Dec 2013, pp. 394–399.

[17] P. Abreu, G. Antonelli, F. Arrichiello, A. Caffaz, A. Caiti, G. Casalino, N. C. Volpi, I. B. De Jong, D. De Palma, H. Duarte *et al.*, "Widely Scalable Mobile Underwater Sonar Technology: An Overview of the H2020 WiMUST project," *Marine Technology Society Journal*, vol. 50, no. 4, pp. 42–53, 2016.

[18] C. Salge and D. Polani, "Empowerment As Replacement for the Three Laws of Robotics," *Frontiers in Robotics and AI*, vol. 4, p. 25, 2017.

[19] C. Guckelsberger, C. Salge, and J. Togelius, "New And Surprising Ways to be Mean: Adversarial NPCs with Coupled Empowerment Minimisation," in *Proc. Conf. CIG*. IEEE, 2018.

[20] C. Salge, C. Glackin, and D. Polani, "Approximation of Empowerment in the Continuous Domain," *Advances in Complex Sys.*, vol. 16, 2012.

[21] N. Ay and D. Polani, "Information Flows in Causal Networks," *Advances in complex systems*, vol. 11, no. 01, pp. 17–41, 2008.

[22] D. J. Soemers, C. F. Sironi, T. Schuster, and M. H. Winands, "Enhancements for real-time monte-carlo tree search in general video game playing," in *Proc. Conf. CIG*. IEEE, 2016, pp. 1–8.

[23] M. Genesereth, N. Love, and B. Pell, "General Game Playing: Overview of the AAAI Competition," *AI Magazine*, vol. 26, no. 2, p. 62, 2005.

[24] D. Perez-Liebana, S. Samothrakis, J. Togelius, S. M. Lucas, and T. Schaul, "General Video Game AI: Competition, Challenges and Opportunities," in *Proc. Conf. AAAI*, 2016, pp. 4335–4337.

Multi-Agent Pathfinding with Real-Time Heuristic Search

Devon Sigurdson
Computing Science
University of Alberta
Edmonton, Canada
dbsigurd@ualberta.ca

Vadim Bulitko
Computing Science
University of Alberta
Edmonton, Canada
bulitko@ualberta.ca

William Yeoh
Computer Science and Engineering
Washington University in St. Louis
St. Louis, USA
wyeoh@wustl.edu

Carlos Hernández
Ciencia de la Ingenieria
Universidad Andres Bello
Santiago, Chile
carlos.hernandez.u@unab.cl

Sven Koenig
Computer Science
University of Southern California
Los Angeles, USA
skoenig@usc.edu

***Abstract*—Multi-agent pathfinding, namely finding collision-free paths for several agents from their given start locations to their given goal locations on a known stationary map, is an important task for non-player characters in video games. A variety of heuristic search algorithms have been developed for this task. Non-real-time algorithms, such as Flow Annotated Replanning (FAR), first find complete paths for all agents and then move the agents along these paths. However, their searches are often too expensive. Real-time algorithms have the ability to produce the next moves for all agents without finding complete paths for them and thus allow the agents to move in real time. Real-time heuristic search algorithms have so far typically been developed for single-agent pathfinding. We, on the other hand, present a real-time heuristic search algorithm for multi-agent pathfinding, called Bounded Multi-Agent A* (BMAA*), that works as follows: Every agent runs an individual real-time heuristic search that updates heuristic values assigned to locations and treats the other agents as (moving) obstacles. Agents do not coordinate with each other, in particular, they neither share their paths nor heuristic values. We show how BMAA* can be enhanced by adding FAR-style flow annotations and allowing agents to push other agents temporarily off their goal locations, when necessary. In our experiments, BMAA* has higher completion rates and lower completion times than FAR.**

***Index Terms*—multi agent pathfinding, video games, artificial intelligence, real-time heuristic search**

I. Introduction

Pathfinding is a core task in many video games, for example, to allow *non-player characters* (NPCs) to move to given goal locations on a known stationary map. A* [1] is a classic algorithm for single-agent pathfinding. The artificial intelligence algorithms in video games, however, often need to find collision-free paths for several agents to their given goal locations. Figure 1 illustrates *multi-agent pathfinding* (MAPF) [2] on a map from the *Dragon Age: Origins* video game [3], where NPCs (green dots) have to move to their given goal locations (red dots).

The constraints on MAPF algorithms depend on the application. For example, real-time strategy games, such as *StarCraft* [4], require the NPCs to move simultaneously in real time, which limits the amount of time available to compute the next moves for all NPCs before they need to start moving. Video games can generate maps procedurally to create new game levels on the fly, which makes it impossible to preprocess the maps. Players can often re-task NPCs at will or the map can change, rendering their previously calculated paths obsolete on a moment's notice. Finally, game settings can limit the amount of coordination allowed among characters in the game (such as sharing their paths or heuristic values), and some characters might not even be under the complete control of the system (because they are on an opposing team).

These constraints motivated our development of *Bounded Multi-Agent A** (BMAA*) — a MAPF algorithm that operates in real time, loses only a small amount of search in case players re-task NPCs or the map changes and neither requires explicit inter-agent coordination, complete control of all NPCs nor preprocessing of maps. BMAA* works as follows: Every agent treats the other agents as (moving) obstacles, runs an individual real-time heuristic search that searches the map around its current location within a given lookahead to select the next move and updates heuristic values assigned to locations to avoid getting stuck. We show how BMAA* can be enhanced by, first, adding flow annotations from the MAPF algorithm FAR [5] (that impose move directions similar to one-way streets) and, second, allowing agents to push other agents temporarily off their goal locations, when necessary, if agents are allowed to send each other move requests. In our experiments, BMAA* has higher completion rates and smaller completion times than FAR, thus demonstrating the promise of real-time heuristic search for MAPF.

II. Problem Formulation

A MAPF problem is defined by a pair (G, A). $G = (N, E, c)$ is an undirected weighted graph of nodes N connected via edges $E \subseteq N \times N$. The costs $c(e)$ of all edges $e \in E$ are strictly positive with the following exceptions: There exists an edge for every node that connects the node

978-1-5386-4359-4/18/$31.00 ©2018 IEEE

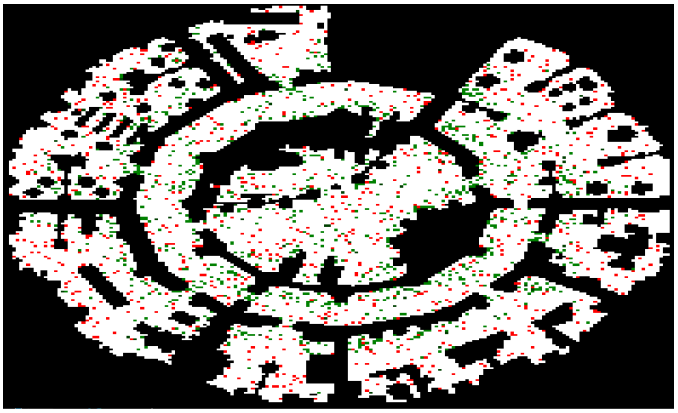


Fig. 1: NPCs on a *Dragon Age: Origins* map [3].

to itself, allowing the agent to always wait in its current node. The costs of these edges are zero. $A = \{a^1, \ldots, a^n\}$ is a set of agents. Each agent $a^i \in A$ is specified by the pair $(n^i_{\text{start}}, n^i_{\text{goal}})$ of its start node n^i_{start} and goal node n^i_{goal}. We use graphs that correspond to rectangular 8-neighbor grids, as is common for video games. The nodes correspond to the cells not blocked by stationary obstacles. The nodes of two neighboring cells are connected with an edge. The costs of these edges are one for cardinal neighbors and $\sqrt{2}$ for diagonal neighbors.

Time advances in discrete steps. Every agent always occupies exactly one node at every time step. We use $n^i_{\text{curr}} \in N$ to refer to the current node of agent a^i. The agent determines a prefix P of a path from its current node to its goal node and sends it to a central NPC controller. $P(n)$ is the successor node of node n on the path. The central NPC controller then moves the agent from node n^i_{curr} to node $P(n^i_{\text{curr}})$ with the following exceptions: The agent waits in its current node if $P(n^i_{\text{curr}})$ is undefined or the agent would collide with another agent. Two agents collide iff they swap nodes or move to the same node from one time step to the next one.

We use the following performance measures: The *completion rate* is the percentage of agents that are in their goal locations when the runtime limit has been reached [5], [6]. The *completion time* for an agent is the time step when that agent last reached its goal location. If an agent leaves its goal and does not return the completion time is undefined. Finally, the *travel distance* of an agent is the sum of the costs of the edges traversed by that agent. We consider the mean of all agents' travel distance and the mean of all agents' completion time as the performance measures in our MAPF problems. These performance measures cannot be optimized simultaneously. Their desired trade-off can be game specific. We choose to maximize the completion rate (because players will notice if NPCs do not reach their goal locations) but report on the other two performance measures as well.

III. Related Work

We now review search algorithms that are related to BMAA*, focusing on pathfinding with heuristic search algorithms, which use heuristic values to focus their search.

A. A*

A* [1] provides the foundation for our BMAA* and many other MAPF algorithms, even though it was developed for single-agent pathfinding. An A* search for an agent explores the search space starting at its current node. The exploration is informed by heuristic values and driven toward nodes with a low estimate of the estimated cost of moving from the current node via them to the goal node. Algorithm 1 shows the pseudo-code for a version of A* that finds a cost-minimal path for agent a^i from its current node n^i_{curr} to its goal node n^i_{goal} under mild assumptions about the graph and the heuristic values.[1] It maintains two lists of nodes, namely the closed and open lists. The closed list is initially empty (line 3), and the open list contains the current node (line 4). The closed list is an unordered set of nodes that A* has already expanded. The open list is an ordered set of nodes that A* considers for expansion. A* always expands a node in the open list with the lowest f-value next, where the f-value of node n is $f(n) = g(n)+h(n)$. Its g-value $g(n)$ is the cost of the lowest-cost path from the current node to node n discovered so far, and its h-value $h(n)$ (or, synonymously, heuristic value) is the heuristic estimate of the cost of a lowest-cost path from node n to the goal node. (The g-values are initially zero for the start node and infinity for all other nodes.) A* removes node n from the open list and adds it to the closed list (lines 11 and 12). It then expands the node by iterating over all of its neighbors n'. It updates the g-value of node n' if node n' has not yet been expanded (i.e., it is not yet in the closed list) and the g-value of node n' can be decreased due to the fact that the cost of the path from the current node via node n to node n' is smaller than the g-value of node n' (because the search has then discovered a lower-cost path from the current node to node n') (line 19). In this case, it also updates the parent of node n' to node n (line 20) and adds it to the open list if it is not already in it (line 22). A* continues its search until either the open list is empty (line 6) or the node about to be expanded is the goal node (line 7). In the former case, no path exists from the current node to the goal node. In the latter case, the path P that is obtained by repeatedly following the parents from the node about to be expanded to the current node is a cost-minimal path from the current node to the goal node in reverse (line 8).

B. Online MAPF Algorithms

We focus on online MAPF algorithms, where there entire problem is not required to be solved before agents begin moving. since we are interested in MAPF algorithms that operate in a short amount of time, lose only a small amount of search in case players re-task NPCs or the map changes and neither require explicit inter-agent coordination, complete

[1] In our pseudo-code, *First* returns a node with the smallest f-value in the open list (breaking ties in favor of a node with the largest g-value, with any remaining ties broken by first-in first-out); *Pop* removes a node with the smallest f-value from the open list (breaking ties in favor of a node with the largest g-value) and returns it; *Add* adds an element to a list; and *GetNeighbors* returns all neighboring nodes of a node in the graph.

Algorithm 1 A*.

```
1:  procedure A*
2:      P ← ()
3:      closed ← ∅
4:      open ← {n^i_curr}
5:      g(n^i_curr) ← 0
6:      while open ≠ ∅ do
7:          if open.First() = n^i_goal then
8:              calculate P
9:              break
10:         end if
11:         n ← open.Pop()
12:         closed.Add(n)
13:         for n′ ∈ n.GetNeighbors() do
14:             if n′ ∉ closed then
15:                 if n′ ∉ open then
16:                     g(n′) ← ∞
17:                 end if
18:                 if g(n′) > g(n) + c(n, n′) then
19:                     g(n′) ← g(n) + c(n, n′)
20:                     n′.parent ← n
21:                     if n′ ∉ open then
22:                         open.Add(n′)
23:                     end if
24:                 end if
25:             end if
26:         end for
27:     end while
```

control of all NPCs nor preprocessing of maps. We describe only the most suitable online MAPF algorithms below.

*Windowed Hierarchical Cooperative A** (WHCA*) [6] finds collision-free paths for all agents for their next window of moves. It shares the paths of all agents up to the given move limit through a reservation table, which adds a time dimension to the search space and thus results in expensive searches. Beyond the move limit, WHCA* simply assumes that every agent follows the cost-minimal path to its goal node and thus ignores collisions among agents. The move limit needs to be sufficiently large to avoid conflicts among agents, resulting in searches that might exceed the amount of time available to compute the next moves for all NPCs before they need to start moving. Furthermore, WHCA* requires all NPCs to be under its complete control.

Flow Annotated Replanning (FAR) [5] combines the reservation table from WHCA* with flow annotations that make its searches less expensive since no time dimension has to be added to the search space. Each agent has to reserve its next moves before it executes them. Agents do not incorporate these reservations into their search but simply wait until other agents that block them have moved away, similar to waiting at traffic lights. FAR attempts to break deadlocks (where several agents wait on each other indefinitely) by pushing some agents temporarily off their goal nodes. However, agents can still get stuck in some cases. The flow annotations of FAR [5] change the edges of the original graph G in order to reduce the number of collisions among agents. They effectively make the undirected original graph directed by imposing move directions on the edges, similar to one-way streets, which reduces the potential for head-to-head collisions among agents. This annotation is done on a grid in a way so that any node remains reachable from all nodes from which it could be reached on the original graph, as follows: The new graph initially has no edges. The first row of nodes is connected via westbound edges, the second row is connected via eastbound edges, and so on. Similarly, the first column of nodes is connected via northbound edges, the second column is connected via southbound edges, and so on. Sink nodes (with only in-bound edges) and source nodes (with only out-bound edges) are handled by adding diagonal edges adjacent to them. If sink and source nodes are in close proximity of each other, the diagonal edges can end up pointing at each other and result in a loss of connectivity, in which case additional undirected edges are added around them. Corridor edges (that is, edges on paths whose interior nodes have degree two) of the original graph remain undirected, which is important in case the corridor is the only connection between two components of the original graph. A standard implementation of A* is then used to search for a path to the goal in this restricted graph.

C. Real-time Heuristic Search

Video games often require NPCs to start moving in a short amount of time, which may not be possible with any of the search algorithms reviewed above since they need to compute a complete path before an agent can execute the first move. *Real-time heuristic search* (RTHS) algorithms, on the other hand, perform a constant amount of search per move regardless of the size of the map or the distance between the start and goal nodes. They have been studied for single-agent pathfinding [7]–[10], starting with the seminal work by Korf [11]. They need to compute only the prefix of a path before the agent can execute the first move — and repeat the operation until the agent reaches the goal node. To avoid cycling forever without reaching the goal node due to the incompleteness of the searches, the algorithms update the heuristic values over time by making them locally consistent [11], incrementally building the open and closed lists [12] or ignoring parts of the map [13]. There are two benefits to using RTHS algorithms in video games. First, an NPC can start moving in a short amount of time. Second, only a small amount of search is lost in case a player re-tasks NPCs or the map changes.

A well-known RTHS algorithm *Real-Time Adaptive A** (RTAA*) [14] performs an A* search, limited to a given number of node expansions. RTAA* then uses the f-value of the node A* was about to expand to update the heuristic values of all expanded nodes (that is, all nodes in the closed list *closed*) as shown in Procedure *Update-Heuristic-Values* in Algorithm 4. The agent then moves along the path from its current node to the node A* was about to expand, limited to a given number of moves — and RTAA* repeats the operation.

IV. Our Approach: BMAA*

Our *Bounded Multi-Agent A** (BMAA*) is a MAPF algorithm where every agent runs its own copy of RTAA*. BMAA* satisfies our requirements: It operates in real-time, loses only a small amount of search in case players re-task NPCs or the map changes. Additionally, it does not requires explicit inter-agent coordination, complete control of all NPCs or preprocessing of maps. The design of BMAA* is modular to allow for extensions by adding or changing modules. For example, BMAA* can be enhanced by, first, adding flow annotations from FAR and, second, allowing agents to push other agents temporarily off their goal nodes, when necessary, if agents are allowed to send each other move requests.

We parameterize BMAA* as follows in the spirit of recent research in the context of *Parameterized Learning Real-Time A** [15]: First, $expansions$ is the limit on the number of node expansions of the A* search of RTAA*. Second, $vision$ is the distance within which agents can see other agents. Third, $moves$ is the number of moves that each agent makes along its path before RTAA* determines a new path for the agent. Fourth, $push$ is a Boolean flag that determines whether agents can push other agents temporarily off their goal nodes. Finally, $flow$ is a Boolean flag that determines whether RTAA* uses the flow annotations from FAR.

A. Procedure NPC-Controller

Algorithm 2 shows the pseudo-code for the central NPC controller. The time step $time$ is initialized to zero at the start of BMAA*, and the central NPC controller is then invoked at every time step with A, the set of agents currently under the control of the system. In the search phase, the central NPC controller lets every agent under the control of the system use the Procedure *Search-Phase* shown in Algorithm 3 to find a prefix of a path from its current node to its goal node (line 3, Algorithm 2). In the subsequent execution phase, the central NPC controller iterates through all agents under the control of the system: First, it retrieves the node that the agent should move to next, which is the successor node of the current node of the agent on its path (line 7, Algorithm 2). Second, if the desired node is blocked by an agent that has reached its own goal node already and agents can push other agents temporarily off their goal nodes ($push = true$), it can push the blocking agent to any neighboring node (line 9, Algorithm 2). The blocking agent returns to its own goal node in subsequent time steps since all agents always execute RTAA* even if they are in their goal nodes. Finally, it moves the agent to the desired node if that node is (no longer) blocked (line 12, Algorithm 2) and increments the current time step (line 16, Algorithm 2).

B. Procedure Search-Phase

Algorithm 3 shows the pseudo-code for the search phase. It finds a new prefix of a path from the current node of the agent to its goal node when it has reached the end of the current path, the current node is no longer on the path (for example, because the agent has been pushed away from its goal node), or the agent has already executed $moves$ moves along the path. (The "expiration" time step $limit$ for the path keeps track of the last condition on line 2 and is set on line 8.) If so, then it uses Procedure *Search* in Algorithm 5 to execute an RTAA* search (line 3) and uses Procedure *Update-Heuristic-Values* to update the heuristic values afterward (lines 5-7).

Algorithm 2 BMAA*'s NPC Controller.

```
1:  procedure NPC-Controller(A)
2:      for all a^i ∈ A do
3:          a^i.Search-Phase()
4:      end for
5:      for all a^i ∈ A do
6:          if a^i.P(n^i_curr) is defined then
7:              n ← a^i.P(n^i_curr)
8:              if push ∧ n is blocked by agent a^j then
9:                  a^j.PushAgent()
10:             end if
11:             if n is not blocked by an agent then
12:                 a^i.MoveTo(n)
13:             end if
14:         end if
15:     end for
16:     time ← time + 1
```

Algorithm 3 BMAA*'s Search Phase.

```
1:  procedure Search-Phase
2:      if Search.P(n^i_curr) is undefined or time ≥ limit then
3:          Search()
4:          if Search.open ≠ ∅ then
5:              n ← Search.open.First()
6:              f ← g(n) + h(n)
7:              Update-Heuristic-Values(Search.closed, f)
8:              limit ← time + moves
9:          end if
10:     end if
```

Algorithm 4 BMAA*'s Update Phase.

```
1:  procedure Update-Heuristic-Values(closed, f)
2:      for n ∈ closed do
3:          h(n) ← f − g(n)
4:      end for
```

C. Procedure Search

Algorithm 5 shows the pseudo-code for an A* search, as discussed before, but with the following changes: First, each agent maintains its own heuristic values across all of its searches. Second, the search also terminates after it has expanded $expansions$ nodes. Thus, the path P obtained on line 9 by repeatedly following the parents from the node about to be expanded to the current node is now only the prefix of a path from the current node of the agent to its goal node. Finally, $GetNeighbors$ returns a node's neighbours that are not blocked by stationary obstacles. However, other agents

Algorithm 5 BMAA*'s Version of A*.

```
1:  procedure SEARCH
2:      P ← ()
3:      exp ← 0
4:      closed ← ∅
5:      open ← {n^i_curr}
6:      g(n^i_curr) ← 0
7:      while open ≠ ∅ do
8:          if open.First() = n^i_goal ∨ exp ≥ expansions then
9:              calculate P
10:             break
11:         end if
12:         n ← open.Pop()
13:         closed.Add(n)
14:         for n′ ∈ n.GetNeighbors(flow) do
15:             d ← distance(n^i_curr, n′)
16:             if n′ is blocked by an agent ∧ d ≤ vision then
17:                 if n′ ≠ n^i_goal then
18:                     continue
19:                 end if
20:             end if
21:             if n′ ∉ closed then
22:                 if n′ ∉ open then
23:                     g(n′) ← ∞
24:                 end if
25:                 if g(n′) > g(n) + c(n, n′) then
26:                     g(n′) ← g(n) + c(n, n′)
27:                     n′.parent ← n
28:                     if n′ ∉ open then
29:                         open.Add(n′)
30:                     end if
31:                 end if
32:             end if
33:         end for
34:         exp ← exp + 1
35:     end while
```

within the straight-line distance *vision* within which agents can see other agents are treated as obstacles as long as they do not block its goal node. Thus, the corresponding nodes are immediately discarded (lines 16-18). If RTAA* uses the flow annotations from FAR ($flow = true$), then *GetNeighbors* returns only those neighboring nodes of a node of the graph which are reachable from the node via the flow annotations from FAR. The flow annotations are not computed in advance but generated the first time the node is processed and then cached so that they can be re-used later.

V. Experimental Evaluation

We experimentally evaluate four versions of BMAA* both against FAR and against A*-Replan, which is equivalent to FAR with no flow annotations. BMAA* cannot push other agents temporarily off their goal locations ($push = false$) and uses no flow annotations ($flow = false$), BMAA*-c can push other agents temporarily off their goal locations, BMAA*-f uses flow annotations, and BMAA*-f-c combines both features. All BMAA* versions use the parameters $lookahead = 32$, $moves = 32$ and $vision = \sqrt{2}$. We choose these parameters on the basis of preliminary experiments. Increasing $lookahead$ often decreases the travel distance at the cost of increasing the search time per move. Increasing $vision$ often reduces the completion rate since it makes agents react to far-away agents. FAR and A*-Replan use a reservation size of three, as suggested by the creators of FAR, meaning that agents must successfully reserve their next three moves before they execute them. All MAPF algorithms use the octile heuristic values as heuristic values (or, in case of BMAA*, to initialize them), are coded in C# and are run on a single Intel Broadwell 2.1Ghz CPU core with 3GB of RAM and a runtime limit of 30 seconds per MAPF instance, which is sufficiently large to allow for full A* searches.

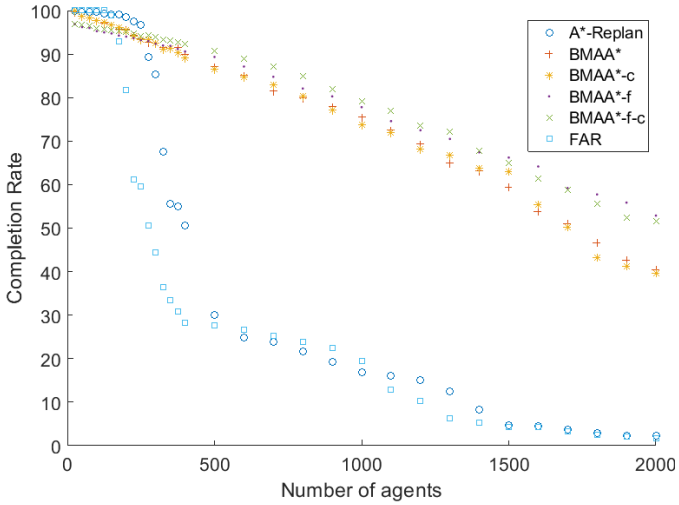


Fig. 2: Completion rates averaged over all MAPF instances.

We evaluate them on ten maps from the MovingAI benchmark set [16]. We use three maps from *Dragon Age: Origins* (DAO), three maps from *WarCraft III* (WCIII), three maps from *Baldur's Gate II* (BGII) (resized to 512×512) and one map from *Baldur's Gate II* in its original size. We create ten MAPF instances for each map with the number of agents ranging from 25 to 400 in increments of 25 and from 400 to 2000 in increments of 200. We assign each agent unique randomly selected start and goal locations which are reachable from each other in the absence of other agents.

A. Aggregate Completion Rate Results

Figure 2 shows the completion rates of all MAPF algorithms averaged over all MAPF instances on all maps. The completion rates of all MAPF algorithms decrease as the number of agents increases because the congestion and amount of search (since every agent has to search) increase. A higher congestion makes it more difficult for agents to reach their goal locations, and a higher amount of search makes it more likely that the runtime limit is reached.

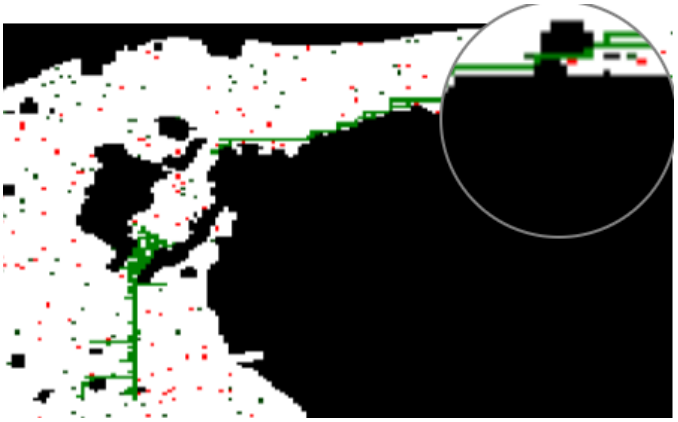


Fig. 3: Issue for FAR: One-cell-wide corridors.

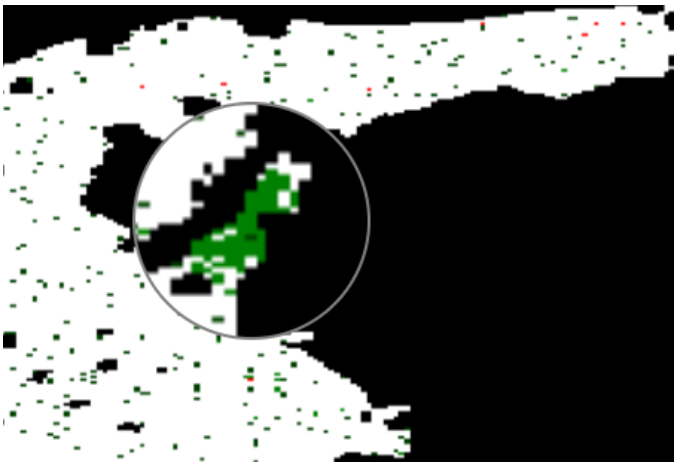


Fig. 4: Issue for BMAA*: Dead ends.

All BMAA* versions have substantially higher completion rates than FAR and A*-Replan for more than 200 agents, with the BMAA* versions that can push other agents temporarily off their goal locations being slightly better than the other BMAA* versions. This can be explained as follows: FAR and A*-Replan determine complete paths for the agents, which results in many agents sharing path segments and thus creates congestion around choke points, such as the one-cell-wide corridor in Figure 3. The BMAA* versions often avoid this behavior, for two reasons: First, the agents of the BMAA* versions have larger travel distances than the ones of FAR and A*-Replan. While the large travel distances of RTHS algorithms are viewed as a major deficiency in the context of single-agent pathfinding, they are beneficial in the context of MAPF since they avoid congestion around choke points. Second, the agents of the BMAA* versions treat the other agents as (moving) obstacles and thus find paths that avoid choke points that are blocked by other agents and thus appear impassable, while the agents of FAR and A*-Replan assume that they can resolve conflicts in their paths with those of other agents and thus move toward choke points.

However, the BMAA* versions also have disadvantages:

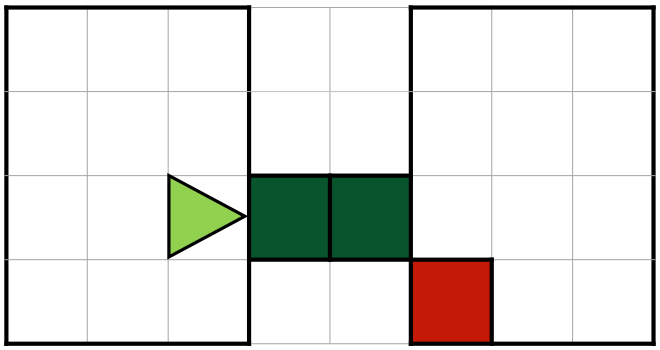


Fig. 5: Unsolvable MAPF instance for the BMAA* versions, where the triangular agent has to move to its red goal location while the dark green square agents are already at their own goal locations in a one-cell-wide corridor.

First, they might move agents into dead ends, such as the one shown in Figure 4, if the initial heuristic values are misleading (resulting in depressions in the heuristic value surface). This well-known issue for RTHS algorithms is addressed by them updating their heuristic values. Several recent RTHS techniques attempt to reduce the travel distances but, of course, agents exploring new areas in imperfect manners could also be viewed as realistic in some cases. Second, even the BMAA* versions that can push other agents temporarily off their goal locations might not be able to move all agents to their goal locations when other agents on their paths are unable to vacate their own goal locations (Figure 5).

B. Per-Map Results

Tables I-III show the three performance measures for all MAPF algorithms averaged over all MAPF instances for each of the maps separately since the map features affect the performance of the MAPF algorithms. The best results are highlighted in bold.

1) Per-Map Completion Rate Results: Table I shows that BMAA*-f-c has the highest completion rates on seven out of the ten maps but the completion rate of BMAA*, for example, is 15 percent larger than the one of BMAA*-f-c on map DAO-lak307d.

2) Per-Map Completion Time Results: The completion rates of FAR and A*-Replan drop substantially for more than 200 agents, as shown in Figure 2. We thus limit the number of agents to 200 since most agents then reach their goal locations. We assign the remaining agents a completion time of 30 seconds. Table II shows that BMAA*-f has the lowest completion times on five maps and BMAA* has the lowest completion times on the remaining four maps.

3) Per-Map Travel Distance Results: We again limit the number of agents to 200 since most agents then reach their goal locations. We assign the remaining agents their travel distances when the runtime limit is reached. Table III shows that FAR has the lowest travel distances on nine maps.

TABLE I: Completion rates averaged over all MAPF instances for each map.

Map Name	A*-Replan	BMAA*	BMAA*-c	BMAA*-f	BMAA*-f-c	FAR	Overall
BGII-AR0414SR (320*281)	45	87	87	85	**89**	32	71
BGII-AR0414SR (512*512)	14	80	79	82	**83**	07	58
BGII-AR0504SR (512*512)	08	51	51	**62**	**62**	05	40
BGII-AR0701SR (512*512)	08	48	49	64	**65**	06	40
WCIII-blastedlands (512*512)	14	**85**	**85**	78	80	03	58
WCIII-duskwood (512*512)	08	58	58	**67**	**67**	03	43
WCIII-golemsinthemist (512*512)	10	59	59	**72**	**72**	04	46
DAO-lak304d (193*193)	19	39	38	**53**	51	27	38
DAO-lak307d (84*84)	60	**79**	77	68	64	60	68
DAO-lgt300d (747*531)	12	65	65	**77**	**77**	10	51
Overall	20	65	65	71	71	16	51

TABLE II: Completion times (in seconds) averaged over all MAPF instances for each map.

Map Name	A*-Replan	BMAA*	BMAA*-c	BMAA*-f	BMAA*-f-c	FAR	Overall
BGII-AR0414SR (320*281)	2.8	**1.2**	5.1	2.2	5.6	3.8	3.5
BGII-AR0414SR (512*512)	8.8	3.6	6.6	**3.0**	6.8	12.9	7.0
BGII-AR0504SR (512*512)	12.3	8.6	12.7	**6.3**	12.5	16.0	11.4
BGII-AR0701SR (512*512)	12.7	4.0	5.4	**3.2**	4.5	15.0	7.5
WCIII-blastedlands (512*512)	8.8	**1.4**	1.5	2.2	2.3	21.0	6.2
WCIII-duskwood (512*512)	12.5	4.1	5.8	**3.7**	5.5	21.1	8.8
WCIII-golemsinthemist (512*512)	11.1	4.2	5.9	**3.0**	4.2	19.0	7.9
DAO-lak304d (193*193)	4.5	6.7	15.1	7.9	11.4	**3.2**	8.1
DAO-lak307d (84*84)	**0.2**	**0.2**	**0.2**	0.5	0.3	0.6	0.3
DAO-lgt300d (747*531)	8.3	**1.4**	1.6	2.2	2.4	10.5	4.4
Overall	8.2	3.5	6.0	**3.4**	5.5	12.3	6.5

TABLE III: Travel distances averaged over all MAPF instances for each map.

Map Name	A*-Replan	BMAA*	BMAA*-c	BMAA*-f	BMAA*-f-c	FAR	Overall
BGII-AR0414SR (320*281)	663	554	557	620	639	**130**	527
BGII-AR0414SR (512*512)	661	1538	1557	2080	2115	**224**	1363
BGII-AR0504SR (512*512)	407	2167	2231	3671	3783	**227**	2089
BGII-AR0701SR (512*512)	562	973	967	1267	1287	**322**	896
WCIII-blastedlands (512*512)	299	376	376	775	784	**268**	480
WCIII-duskwood (512*512)	367	1179	1188	1712	1737	**257**	1073
WCIII-golemsinthemist (512*512)	530	1205	1206	1371	1369	**285**	994
DAO-lak304d (193*193)	2154	1425	1460	1258	1295	**148**	1290
DAO-lak307d (84*84)	578	**38**	39	125	95	47	154
DAO-lgt300d (747*531)	435	403	404	592	603	**289**	454
Overall	666	986	998	1347	1371	**225**	932

VI. Conclusions

Our paper considered an important problem faced by artificial intelligence in many video games, namely MAPF. We reviewed recent related work and argued for the use of real-time heuristic search. We then contributed a new real-time MAPF algorithm, BMAA*, which is of modular design and can be enhanced with recent flow-annotation techniques. BMAA* has higher completion rates and smaller completion times than FAR at the cost of longer travel distances, which is a good trade-off since NPCs reaching their goal locations via possibly longer paths is less noticeable by players than NPCs not reaching their goal locations at all. Finally, we discussed what makes MAPF difficult for different algorithms, paving the road to per-problem algorithm selection techniques in the spirit of recent research in the context of single-agent pathfinding [17], [18].

Overall, BMAA* demonstrates the promise of real-time heuristic search for MAPF. Its main shortcoming is its large travel distances compared to the ones of FAR. Several recent RTHS techniques attempt to reduce the travel distances for single-agent pathfinding [19] and thus might also be able to reduce the travel distances for BMAA*. Examples include search space reduction techniques [13], [20], precomputation techniques [21], [22] and initialization techniques for the heuristic values, which might help to reduce the dead-end problem shown in Figure 4.

Acknowledgments

Devon Sigurdson and Vadim Bulitko appreciate support from NSERC and Nvidia. Carlos Hernández was partially funded by Fondecyt grant number 1161526. Sven Koenig was supported by the National Science Foundation (NSF) under grant numbers 1724392, 1409987 and 1319966 as well as a

gift from Amazon. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the sponsoring organizations, agencies or the U.S. government.

REFERENCES

[1] P. Hart, N. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[2] S. Koenig and H. Ma, "AI buzzwords explained: Multi-agent path finding (MAPF)," *AI Matters*, 2017.

[3] BioWare, "Dragon Age: Origins," 2009.

[4] "Starcraft," https://starcraft2.com/en-us/, accessed: 2018-03-14.

[5] K.-H. C. Wang and A. Botea, "Fast and memory-efficient multi-agent pathfinding," in *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 2008, pp. 380–387.

[6] D. Silver, "Cooperative pathfinding," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 2005, pp. 117–122.

[7] T. Ishida, *Real-time search for learning autonomous agents*. Springer Science & Business Media, 1997, vol. 406.

[8] V. Bulitko and G. Lee, "Learning in real time search: A unifying framework," *Journal of Artificial Intelligence Research*, vol. 25, pp. 119–157, 2006.

[9] S. Koenig and X. Sun, "Comparing real-time and incremental heuristic search for real-time situated agents," *Journal of Autonomous Agents and Multi-Agent Systems*, vol. 18, no. 3, pp. 313–341, 2009.

[10] B. Cserna, M. Bogochow, S. Chambers, M. Tremblay, S. Katt, and W. Ruml, "Anytime versus real-time heuristic search for on-line planning," in *Proceedings of the Symposium on Combinatorial Search (SoCS)*, 2016.

[11] R. Korf, "Real-time heuristic search," *Artificial Intelligence*, vol. 42, no. 2–3, pp. 189–211, 1990.

[12] Y. Björnsson, V. Bulitko, and N. Sturtevant, "TBA*: Time-bounded A*," in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2009, pp. 431–436.

[13] C. Hernandez, A. Botea, J. A. Baier, and V. Bulitko, "Online bridged pruning for real-time search with arbitrary lookaheads," in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2017, pp. 510–516.

[14] S. Koenig and M. Likhachev, "Real-time Adaptive A*," in *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2006, pp. 281–288.

[15] V. Bulitko, "Evolving real-time heuristic search algorithms," in *Proceedings of the International Conference on the Synthesis and Simulation of Living Systems*, 2016.

[16] N. R. Sturtevant, "Benchmarks for grid-based pathfinding," *Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 2, pp. 144–148, 2012.

[17] V. Bulitko, "Per-map algorithm selection in real-time heuristic search," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 2016, pp. 143–148.

[18] D. Sigurdson and V. Bulitko, "Deep learning for real-time heuristic search algorithm selection," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 2017, pp. 108–114.

[19] V. Bulitko and A. Sampley, "Weighted lateral learning in real-time heuristic search," in *Proceedings of the Symposium on Combinatorial Search (SoCS)*, 2016.

[20] C. Hernandez and J. A. Baier, "Avoiding and escaping depressions in real-time heuristic search," *Journal of Artificial Intelligence Research*, vol. 43, pp. 523–570, 2012.

[21] R. Lawrence and V. Bulitko, "Database-driven real-time heuristic search in video-game pathfinding," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 5, no. 3, pp. 227–241, 2013.

[22] L. Cohen, T. Uras, T. Kumar, H. Xu, N. Ayanian, and S. Koenig, "Improved solvers for bounded-suboptimal multi-agent path finding," in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2016, pp. 3067–3074.

Bayesian Opponent Exploitation in Imperfect-Information Games

Sam Ganzfried
Ganzfried Research
Miami Beach, Florida, 33139
sam@ganzfriedresearch.com
School of Computing and Information Sciences
Florida International University
Miami, Florida, 33199
sganzfri@cis.fiu.edu

Qingyun Sun
Department of Mathematics
Stanford University
Stanford, CA, 94305
qysun@stanford.edu

***Abstract*—Two fundamental problems in computational game theory are computing a Nash equilibrium and learning to exploit opponents given observations of their play (opponent exploitation). The latter is perhaps even more important than the former: Nash equilibrium does not have a compelling theoretical justification in game classes other than two-player zero-sum, and for all games one can potentially do better by exploiting perceived weaknesses of the opponent than by following a static equilibrium strategy throughout the match. The natural setting for opponent exploitation is the Bayesian setting where we have a prior model that is integrated with observations to create a posterior opponent model that we respond to. The most natural, and a well-studied prior distribution is the Dirichlet distribution. An exact polynomial-time algorithm is known for best-responding to the posterior distribution for an opponent assuming a Dirichlet prior with multinomial sampling in normal-form games; however, for imperfect-information games the best known algorithm is based on approximating an infinite integral without theoretical guarantees. We present the first exact algorithm for a natural class of imperfect-information games. We demonstrate that our algorithm runs quickly in practice and outperforms the best prior approaches. We also present an algorithm for a uniform prior.**

***Index Terms*—game theory; opponent modeling; imperfect information; Dirichlet prior; uniform prior; Bayesian approach**

I. Introduction

Imagine you are playing a game repeatedly against one or more opponents. What algorithm should you use to maximize your performance? The classic "solution concept" in game theory is the Nash equilibrium. In a Nash equilibrium σ, each player is simultaneously maximizing his payoff assuming the opponents all follow their components of σ. So should we just find a Nash equilibrium strategy for ourselves and play it in all the game iterations?

Unfortunately, there are some complications. First, there can exist many Nash equilibria, and if the opponents are not following the same one that we have found (or are not following one at all), then our strategy would have no performance guarantees. Second, finding a Nash equilibrium is challenging computationally: it is PPAD-hard and is widely conjectured that no polynomial-time algorithms exist [1]. These challenges apply to both extensive-form games (of both perfect and imperfect information) and strategic-form games, for games with more than two players and non-zero-sum games. While a particular Nash equilibrium may happen to perform well in practice,[1] there is no theoretically compelling justification for why computing one and playing it repeatedly is a good approach. Two-player zero-sum games do not face these challenges: there exist polynomial-time algorithms for computing an equilibrium [3], and there exists a game value that is guaranteed in expectation in the worst case by all equilibrium strategies regardless of the strategy played by the opponent (and this value is the best worst-case guaranteed payoff for any of our strategies). However, even for this game class it would be desirable to deviate from equilibrium to learn and exploit perceived weaknesses of the opponent; for instance, if the opponent has played Rock in each of the first 500 iterations of rock-paper-scissors, it seems desirable to put additional weight on paper beyond the equilibrium value of $\frac{1}{3}$.

Thus, learning to exploit opponents' weaknesses is desirable in all game classes. One approach would be to construct an opponent model consisting of a single mixed strategy that we believe the opponent is playing given our observations of his play and a prior distribution (perhaps computed from a database of historical play). This approach has been successfully applied to exploit weak agents in limit Texas hold 'em poker, a large imperfect-information game [4].[2] A drawback is that it is potentially not robust. It is very unlikely that the opponent's strategy matches this point estimate exactly, and we could perform poorly if our model is incorrect. A more robust approach, which is the natural one to use in this setting, is to use a Bayesian model, where the prior and posterior are full distributions over mixed strategies of the opponent, not single mixed strategies. A natural prior distribution, which has been

[1] An agent for 3-player limit Texas hold 'em computed by the counterfactual regret minimization algorithm (which converges to Nash equilibrium in certain games) performed well in practice despite a lack of theoretical justification [2].

[2] This approach used an approximate Nash equilibrium strategy as the prior and is applicable even when historical data is not available, though if additional data were available a more informed prior that capitalizes on the data would be preferable.

978-1-5386-4359-4/18/$31.00 ©2018 IEEE

studied and applied in this context, is the Dirichlet distribution. The pdf of the Dirichlet distribution is the belief that the probabilities of K rival events are x_i given that each event has been observed $\alpha_i - 1$ times: $f(x, \alpha) = \frac{1}{B(\alpha)} \prod x_i^{\alpha_i - 1}$.[3] Some notable properties are that the mean is $E[X_i] = \frac{\alpha_i}{\sum_k \alpha_k}$ and that, assuming multinomial sampling, the posterior after including new observations is also Dirichlet, with parameters updated based on the new observations.

Prior work has presented an efficient algorithm for optimally exploiting an opponent in normal-form games in the Bayesian setting with a Dirichlet prior [5], which is essentially the fictitious play rule [6]. Given prior counts α_i for each opponent action, the algorithm increments the counter for an action by one each time it is observed, and then best responds to a model for the opponent where he plays each strategy in proportion to the counters. This algorithm would also extend directly to sequential games of perfect information, where we maintain independent counters at each opponent decision node; this would also work for games of imperfect information where the opponent's private information is observed after each round (so that we would know exactly what information set he took the observed action from). For all of these game classes the algorithm would apply to both zero and general-sum games, for any number of players. However, it would not apply to imperfect-information games where opponents' private information is not observed after play.

An algorithm exists for approximating a Bayesian best response in imperfect-information games, which uses importance sampling to approximate an infinite integral. This algorithm has been successfully applied to limit Texas hold 'em poker [7]. [4] However, it is only a heuristic approach with no guarantees. The authors state,

> "Computing the integral over opponent strategies depends on the form of the prior but is difficult in any event. For Dirichlet priors, it is possible to compute the posterior exactly but the calculation is expensive except for small games with relatively few observations. This makes the exact BBR an ideal goal rather than a practical approach. For real play, we must consider approximations to BBR."

However, we see no justification for the claim that it is possible to compute the posterior exactly in prior work, and there could easily be no closed-form solution. In this paper we present a solution for this problem, leading to the first exact optimal algorithm for performing Bayesian opponent exploitation in imperfect-information games. While the claim is correct that the computation is expensive for large games, we show that in a small yet realistic game it outperforms all prior approaches, which are based on sampling. Furthermore, we show that the computation can run extremely quickly even for large number of observations (though it can run into numerical instability). We also present general theory and an algorithm for another natural prior distribution (uniform over a polyhedron).

[3] $B(\alpha)$ is the beta function $B(\alpha) = \frac{\prod \Gamma(\alpha_i)}{\Gamma(\sum_i \alpha_i)}$, where $\Gamma(n) = (n-1)!$ is the gamma function.

[4] In addition to Bayesian Best Response, the paper also considers heuristic approaches for approximating several other response functions: Max A Posteriori Response and Thompson's Response.

II. META-ALGORITHM

The problem of developing efficient algorithms for optimizing against a posterior distribution, which is a probability distribution over mixed strategies for the opponent (which are themselves distributions over pure strategies) seems daunting. We need to be able to compactly represent the posterior distribution and efficiently compute a best response to it. Fortunately, we show that our payoff of playing any strategy σ_i against a probability distribution over mixed strategies for the opponent equals our payoff of playing σ_i against the mean of the distribution. Thus, we need only represent and respond to the single strategy that is the mean of the distribution, and not to the full distribution. While this result was likely known previously, we have not seen it stated explicitly, and it is important enough to be highlighted so that it is on the radar of the AI community.

Suppose the opponent is playing mixed strategy σ_{-i} where $\sigma_{-i}(s_{-j})$ is the probability that he plays pure strategy $s_{-j} \in S_{-j}$. By definition of expected utility, $u_i(\sigma_i, \sigma_{-i}) = \sum_{s_{-j} \in S_{-j}} \sigma_{-i}(s_{-j}) u_i(\sigma_i, s_{-j})$. We can generalize this naturally to the case where the opponent is playing according to a probability distribution with pdf f_{-i} over mixed strategies: $u_i(\sigma_i, f_{-i}) = \int_{\sigma_{-i} \in \Sigma_{-i}} \left[f_{-i}(\sigma_{-i}) \cdot u_i(\sigma_i, \sigma_{-i}) \right]$. Let $\overline{f_{-i}}$ denote the mean of f_{-i}. That is, $\overline{f_{-i}}$ is the mixed strategy that selects s_{-j} with probability $\int_{\sigma_{-i} \in \Sigma_{-i}} \left[\sigma_{-i}(s_{-j}) \cdot f_{-i}(\sigma_{-i}) \right]$. Then we have the following:

Theorem 1.

$$u_i(\sigma_i, \overline{f_{-i}}) = u_i(\sigma_i, f_{-i}).$$

That is, the payoff against the mean of a strategy distribution equals the payoff against the full distribution.

Proof.

$$\begin{aligned}
& u_i(\sigma_i, \overline{f_{-i}}) \\
= & \sum_{s_{-j} \in S_{-j}} \left[u_i(\sigma_i, s_{-j}) \int_{\sigma_{-i} \in \Sigma_{-i}} \left[\sigma_{-i}(s_{-j}) \cdot f_{-i}(\sigma_{-i}) \right] \right] \\
= & \sum_{s_{-j} \in S_{-j}} \left[\int_{\sigma_{-i} \in \Sigma_{-i}} \left[u_i(\sigma_i, s_{-j}) \cdot \sigma_{-i}(s_{-j}) \cdot f_{-i}(\sigma_{-i}) \right] \right] \\
= & \int_{\sigma_{-i} \in \Sigma_{-i}} \left[\sum_{j \in S_{-j}} \left[u_i(\sigma_i, s_{-j}) \cdot \sigma_{-i}(s_{-j}) \cdot f_{-i}(\sigma_{-i}) \right] \right] \\
= & \int_{\sigma_{-i} \in \Sigma_{-i}} \left[u_i(\sigma_i, \sigma_{-i}) \cdot f_{-i}(\sigma_{-i}) \right] \\
= & u_i(\sigma_i, f_{-i})
\end{aligned}$$

□

Theorem 1 applies to normal and extensive-form games (with perfect or imperfect information), for any number of players (σ_{-i} could be a joint strategy profile for all opponents).

Now suppose the opponent is playing according a prior distribution $p(\sigma_{-i})$, and let $p(\sigma_{-i}|x)$ denote the posterior

probability given observations x. Let $\overline{p(\sigma_{-i}|x)}$ denote the mean of $p(\sigma_{-i}|x)$. As an immediate consequence of Theorem 1, we have the following corollary.

Corollary 1. $u_i(\sigma_i, \overline{p(\sigma_{-i}|x)}) = u_i(\sigma_i, p(\sigma_{-i}|x))$.

Corollary 1 implies the meta-procedure for optimizing performance against an opponent using p:

Algorithm 1 Meta-algorithm for Bayesian opponent exploitation

Inputs: Prior distribution p_0, response functions r_t

$M_0 \leftarrow \overline{p_0(\sigma_{-i})}$
$R_0 \leftarrow r_0(M_0)$
Play according to R_0
for $t = 1$ to T **do**
  $x_t \leftarrow$ observations of opponent's play at time step t
  $p_t \leftarrow$ posterior distribution of opponent's strategy given prior p_{t-1} and observations x_t
  $M_t \leftarrow$ mean of p_t
  $R_t \leftarrow r_t(M_t)$
  Play according to R_t

There are several challenges for applying Algorithm 1. First, it assumes that we can compactly represent the prior and posterior distributions p_t, which have infinite domain (the set of opponents' mixed strategy profiles). Second, it requires a procedure to efficiently compute the posterior distributions given the prior and the observations, which requires updating potentially infinitely many strategies. Third, it requires an efficient procedure to compute the mean of p_t. And fourth, it requires that the full posterior distribution from one round be compactly represented to be used as the prior in the next round. We can address the fourth challenge by using a modified update step:

$p_t \leftarrow$ posterior distribution of opponent's strategy given prior p_0 and observations $x_1, \ldots, x_t$.

We will be using this new rule in our main algorithm.

The response functions r_t (which return a strategy for ourselves that performs well against input opponents' strategies) could be standard best response, for which linear-time algorithms exist in games of imperfect information (and a recent approach has enabled efficient computation in extremely large games [8]). They could also be a more robust response, e.g., one that places a limit on the exploitability of our own strategy, perhaps one that varies over time based on performance (or a lower-variance estimator) [9], [10], [11]. In particular, the restricted Nash response has been demonstrated to outperform best response against agents in limit Texas hold 'em whose actual strategy may differ substantially from the exact model [9].

III. Robustness of the approach

It has been pointed out that, empirically, the approach described is not robust: if we play a full best response to a point estimate of the opponent's strategy we can have very high exploitability ourselves, and could perform very poorly if in fact we are wrong about our model [9]. This could happen for several reasons. Our modeling algorithm could be incorrect: it could make an incorrect assumption about the prior and form of the opponent's distribution. This could happen for several reasons. One reason is that the opponent could actually be changing his strategy over time (possibly either by improving his own play or by adapting to our play), in which case a model that assumes a static opponent could be predicting a strategy that the opponent is no longer using. The opponent could also have modified his play strategically in an attempt to deceive us by playing one way initially and then counter-exploiting us as we attempt to exploit the model we have formed from his initial strategy (e.g., the opponent initially starts off playing extremely conservatively, then switches to a more aggressive style as he suspects we will start to exploit his extreme conservatism). His initial strategy need not arise from deception: it is also possible that simply due to chance events (either due to his own randomization in his strategy or due to the states of private information selected by chance) the opponent has appeared to be playing in a certain way (e.g., very conservatively), and as he becomes aware of this conservative "image," naturally it occurs to him to modify his play by becoming more aggressive.

A second reason that we could be wrong in our opponent model other than our algorithm incorrectly modeling the opponents' dynamic approach is that our observations of his play are very noisy (due to both randomization in the opponent's strategy and to the private information selected by chance), particularly over a small sample. Even if our approach is correct and the opponent is in fact playing a static strategy according to the distribution assumed by the modeling algorithm, it is very unlikely that our actual perception of his strategy is precisely correct.

A third reason, of course, is that the opponent may be following a static strategy that does not exactly conform to our model for the prior and/or sampling method used to generate the posterior.

We would like an approach that is robust in the event that our model of the opponent's strategy is incorrect, whichever the cause may be. Prior work has considered a model where the opponent plays according to a model x_{-i} with probability p and with probability $1-p$ plays a nemesis to our strategy [9]. For carefully selected values of p (typically 0.95 or 0.99), they show that this can achieve a relatively high level of exploitation (similar to a full best response) with a significantly smaller worst-case exploitability. We note that, as described in Section II, Algorithm 1 can be integrated with any response function, not necessarily a full best response, and so r_t could be selected to be the Restricted Nash Response from prior work [9]. However, it seems excessively conservative to give the opponent credit for playing a full nemesis to our strategy; if we are relatively confident in our opponent model, then a more reasonable robustness criterion would be to explore performance as we allow the opponent's strategy to differ by a small amount from the predicted strategy (i.e., the opponent

is playing a strategy that is very close to our model, and not necessarily putting weight on a full nemesis).

Suppose we believe the opponent is playing x_{-i}, while he is actually playing x'_{-i}. Let M be the maximum absolute value of a utility to player i, and let N be the maximum number of actions available to a player. Let $\epsilon > 0$ be arbitrary. Then, if $|x_{-i}(j) - x'_{-i}(j)| < \delta$ for all j, where $\delta = \frac{\epsilon}{MN}$,

$$\begin{aligned}
& |u_i(\sigma^*, x_{-i}) - u_i(\sigma^*, x'_{-i})| \\
= & \left| \sum_j (x_{-i}(j) - x'_{-i}(j)) u_i(\sigma^*, s_{-j}) \right| \\
<= & \sum_j \left| \left(x_{-i}(j) - x'_{-i}(j)\right) u_i(\sigma^*, s_{-j}) \right| \\
<= & \sum_j \left(\left|x_{-i}(j) - x'_{-i}(j)\right| \cdot |u_i(\sigma^*, s_{-j})| \right) \\
<= & \sum_j \left(|x_{-i}(j) - x'_{-i}(j)| \cdot M \right) \\
< & M \sum_j \delta <= MN\delta = MN \cdot \frac{\epsilon}{MN} = \epsilon
\end{aligned}$$

This same analysis can be applied directly to show that our payoff is continuous in the opponent's strategy for many popular distance functions (i.e., for any distance function where one strategy can get arbitrarily close to another as the components get arbitrarily close). For instance this would apply to L1, L2, and earth mover's distance, which have been applied previously to compute distances been strategies within opponent exploitation algorithms [4]. Thus, if we are slightly off in our model of the opponent's strategy, even if we are doing a full best response we will do only slightly worse.

IV. Exploitation algorithm for Dirichlet prior

As described in Section I the Dirichlet distribution is the conjugate prior for the multinomial distribution, and therefore the posterior is also a Dirichlet distribution, with the parameters α_i updated to reflect the new observations. Thus, the mean of the posterior can be computed efficiently by computing the strategy for the opponent in which he plays each strategy in proportion to the updated weight, and Algorithm 1 yields an exact efficient algorithm for computing the Bayesian best response in normal-form games with a Dirichlet prior. However, the algorithm does not apply to games of imperfect information since we do not observe the private information held by the opponent, and therefore do not know which of his action counters we should increment. In this section we will present a new algorithm for this setting. We present it in the context of a representative motivating game where the opponent is dealt a state of private information and then takes publicly-observable action, and present the algorithm for the general setting in Section IV-C.

We are interested in studying the following two-player game setting. Player 1 is given private information state x_i (according to a probability distribution). Then he takes a publicly observable action a_i. Player 2 then takes an action after observing player 1's action (but not his private information), and both players receive a payoff. We are interested in player 2's problem of inferring the (assumed stationary) strategy of player 1 after repeated observations of the public action taken (but not the private information). Note that this setting is very general. For example, in poker x_i could denote the opponent's private card(s) and a_i denote the amount he bets, and in an ad auction x_i could denote his valuation (e.g., high or low), and a_i could denote the amount he bids [12].

A. Motivating game and algorithm

For concreteness and motivation, consider the following poker game instantiation of this setting, where we play the role of player 2. Let's assume that in this two-player game, player 1 is dealt a King (K) and Jack (J) with probability $\frac{1}{2}$, while player 2 is always dealt a Queen. Player 1 is allowed to make a big bet of \$10 (b) or a small bet of \$1 (s), and player 2 is allowed to call or fold. If player 2 folds, then player 1 wins the \$2 pot (for a profit of \$1); if player 1 bets and player 2 calls then the player with the higher card wins the \$2 pot plus the size of the bet.

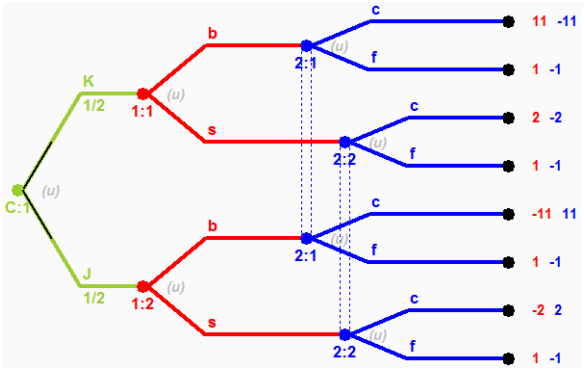


Fig. 1. Chance deals player 1 king or jack with probability $\frac{1}{2}$ at the green node. Then player 1 selects big or small bet at a red node. Then player 2 chooses call or fold at a blue node.

If we observe player 1's card after each hand, then we can apply the approach described above, where we maintain a counter for player 1 choosing each action with each card that is incremented for the selected action. However, if we do not observe player 1's card after the hand (e.g., if we fold), then we would not know whether to increment the counter for the king or the jack. To simplify analysis, we will assume that we never observe the opponent's private card after the hand (which is not realistic since we would observe his card if he bets and we call); we can assume that we do not observe our payoff either until all game iterations are complete, since that could allow us to draw inferences about the opponent's card. There are no known algorithms even for the simplified case of fully unobservable opponent's private information. We suspect that an algorithm for the case when the opponent's private information is sometimes observed can be constructed based on our algorithm, and we plan to study this problem in future work.

From analysis in the accompanying tech report [13], we are able to compute a closed-form expression for the expectation of the posterior probability that the opponent takes action b with a Jack given that we have just observed him take action b (the other quantities can be computed analogously), which is denoted by $P(b|O, J)$.

$$\frac{B(\alpha_{Kb}+1,\alpha_{Ks})B(\alpha_{Jb}+1,\alpha_{Js})+B(\alpha_{Kb},\alpha_{Ks})B(\alpha_{Jb}+2,\alpha_{Js})}{Z} \quad (1)$$

where the denominator Z is equal to

$$B(\alpha_{Kb}+1,\alpha_{Ks})B(\alpha_{Jb}+1,\alpha_{Js})+B(\alpha_{Kb},\alpha_{Ks})B(\alpha_{Jb}+2,\alpha_{Js})$$
$$+B(\alpha_{Kb}+1,\alpha_{Ks})B(\alpha_{Jb},\alpha_{Js}+1)+B(\alpha_{Kb},\alpha_{Ks})B(\alpha_{Jb}+1,\alpha_{Js}+1).$$

Note that the algorithm we have presented applies for the case where we play one more game iteration and collect one additional observation. However, it is problematic for the general case we are interested in where we play many game iterations, since the posterior distribution is not Dirichlet, and therefore we cannot just apply the same procedure in the next iteration using the computed posterior as the new prior. We will need to derive a new expression for $P(b|O, J)$ for this setting. Suppose that we have observed the opponent play action b for θ_b times and s θ_s times (in addition to the number of fictitious observations reflected in the prior α), though we do not observe his card. Then $P(b|O, J)$ equals

$$\frac{\sum_{i=0}^{\theta_b}\sum_{j=0}^{\theta_s}B(\alpha_{Kb}+i,\alpha_{Ks}+j)B(\alpha_{Jb}+\theta_b-i+1,\alpha_{Js}+\theta_s-j)}{Z} \quad (2)$$

The normalization term is

$$Z=\sum_i\sum_j[B(\alpha_{Kb}+i,\alpha_{Ks}+j)B(\alpha_{Jb}+\theta_b-i+1,\alpha_{Js}+\theta_s-j)$$
$$+B(\alpha_{Kb}+i,\alpha_{Ks}+j)B(\alpha_{Jb}+\theta_b-i,\alpha_{Js}+\theta_s-j+1)].$$

Details of the derivation are in the tech report.

Thus the algorithm for responding to the opponent is the following. We start with the prior counters on each private information-action combination, α_{Kb}, α_{Ks}, etc. We keep separate counters θ_b, θ_s for the number of times we have observed each action during play. Then we combine these counters according to Equation 2 in order to compute the strategy for the opponent that is the mean of the posterior given the prior and observations, and we best respond to this strategy, which gives us the same payoff as best responding to the full posterior distribution according to Theorem 1. There are only O(n^2) terms in the expression given by Equation 2, so this algorithm is efficient.

B. Example

Suppose the prior is that the opponent played b with K 10 times, played s with K 3 times, played b with J 4 times, and played s with J 9 times. Thus $\alpha_{Kb} = 10, \alpha_{Ks} = 3, \alpha_{Jb} = 4, \alpha_{Js} = 9$. Now suppose we observe him play b at the next iteration. Applying our algorithm using Equation 1 gives

$$p(b|O,J)=\frac{B(11,3)B(5,9)+B(10,3)(6,9)}{Z}=\frac{2.65209525e^{-7}}{Z}$$

$$p(s|O,J)=\frac{B(11,3)B(4,10)+B(10,3)(5,10)}{Z}=\frac{5.5888056e^{-7}}{Z}$$

$$\longrightarrow p(b|O,J)=\frac{2.65209525e^{-7}}{2.65209525e^{-7}+5.5888056e^{-7}}=0.3218210361.$$

So we think that with a jack he is playing a strategy that bets big with probability 0.322 and small with probability 0.678. Notice that previously we thought his probability of betting big with a jack was $\frac{4}{13} = 0.308$, and had we been in the setting where we always observe his card after gameplay and observed that he had a jack, the posterior probability would be $\frac{5}{14} = 0.357$.

An alternative "naïve" (and incorrect) approach would be to increment α_{Jb} by $\frac{\alpha_{Jb}}{\alpha_{Jb}+\alpha_{Kb}}$, the ratio of the prior probability that he bets big given J to the total prior probability that he bets big. This gives a posterior probability of him betting big with J of $\frac{4+\frac{4}{13}}{14} = 0.308$, which differs significantly from the correct value. It turns out that this approach is actually equivalent to just using the prior:

$$\frac{x+\frac{x}{x+y}}{x+y+1}\cdot\frac{x+y}{x+y}=\frac{x(x+y)+x}{(x+y+1)(x+y)}=\frac{x}{x+y}$$

C. Algorithm for general setting

We now consider the general setting where the opponent can have n different states of private information according to an arbitrary distribution π and can take m different actions. Assume he is given private information x_i with probability π_i, for $i = 1, \ldots, n$, and can take action k_i, for $i = 1, \ldots, m$. Assume the prior is Dirichlet with parameters α_{ij} for the number of times action j was played with private information i (so the mean of the prior has the player selecting action k_j at state x_i with probability $\frac{\alpha_{ij}}{\sum_j \alpha_{ij}}$). Assume that action k_{j^*} was observed in a new time step, while the opponent's private information was not observed. We now compute the expectation for the posterior probability that the opponent plays k_{j^*} with private information x_{i^*}.

$$\begin{aligned}
& P(A=k_{j^*}|O,C=x_{i^*}) \\
= & \frac{\int\left[q_{k_j^*|x_i^*}\sum_{i=1}^n\left[\pi_i q_{k_{j^*}|x_i}\prod_{h=1}^m\prod_{j=1}^n q_{k_h|x_j}^{\alpha_{jh}-1}\right]\right]}{p(O)\prod_{i=1}^n B(\alpha_{i1},\ldots,\alpha_{im})} \\
= & \frac{\sum_i\left[\pi_i\prod_j B(\gamma_{1j},\ldots,\gamma_{nj})\right]}{Z},
\end{aligned}$$

where $\gamma_{ij} = \alpha_{ij} + 2$ if $i = i^*$ and $j = j^*$, $\gamma_{ij} = \alpha_{ij} + 1$ if $j = j^*$ and $i \neq i^*$, and $\gamma_{ij} = \alpha_{ij}$ otherwise. If we denote the numerator by $\tau_{i^*j^*}$ then $Z = \sum_{i^*}\tau_{i^*j^*}$. Notice that the product is over n terms, and therefore the total number of terms will be exponential in n (it is O($m \cdot 2^n$)).

For the case of multiple observed actions, the posterior is not Dirichlet and cannot be used directly as the prior for the next iteration. Suppose we have observed action k_j θ_j times (in addition to the number of fictitious times indicated by the prior counts α_{ij}). We compute $P(q|O)$ analogously as

$$P(q|O) = \frac{\sum_{i=1}^{n}\left[\pi_i \sum_{\{\rho_{ab}\}} \prod_{h=1}^{m} \prod_{j=1}^{n} q_{k_h|x_j}^{\alpha_{jh}-1+\rho_{jh}}\right]}{p(O)\prod_{i=1}^{n} B(\alpha_{i1},\ldots,\alpha_{im})},$$

where the $\sum_{\{\rho_{ab}\}}$ is over all values $0 \leq \rho_{ab} \leq \theta_b$ with $\sum_a \rho_{ab} = \theta_b$ for each b, for $1 \leq a \leq n$, $1 \leq b \leq m$:

$$\sum_{\{\rho_{ab}\}} = \sum_{\rho_{1b}=0}^{\theta_b} \sum_{\rho_{2b}=0}^{\theta_b-\rho_{1b}} \cdots \sum_{\rho_{n-1,b}=0}^{\theta_b-\sum_{r=0}^{n-2}\rho_{rb}} \sum_{\rho_{nb}=\theta_b-\sum_{r=0}^{n-2}\rho_{rb}}^{\theta_b-\sum_{r=0}^{n-1}\rho_{rb}}.$$

The expression for the full posterior distribution is

$$P(q|O) = \frac{\sum_i \left[\pi_i \sum_{\{\rho_{ab}\}} \prod_h B(\alpha_{1h}+\rho_{1h},\ldots,\alpha_{nh}+\rho_{nh})\right]}{Z}$$

The total number of terms is $O\left(\left(\frac{(T+n)!}{n!T!}\right)^m\right)$, which is exponential in the number of private information states and actions, but polynomial in the number of iterations.

The following theorem shows an approach for computing products of the beta function that leads to an exponential improvement in the running time of the algorithm for one observation, and reduces the dependence on m for the multiple observation setting from exponential to linear, though the complexity still remains exponential in n and T for the latter. See tech report for full details [13].

Theorem 2. *Define $\gamma_j = \sum_{i=1}^{n} \gamma_{ij}$ and the empirical probability distribution $\hat{P}_j(i) = \frac{\gamma_{ij}}{\sum_{i=1}^{n}\gamma_{ij}} = \frac{\gamma_{ij}}{\gamma_j}$. Define the Gamma function $\Gamma(x) = \int_0^\infty x^{z-1}e^{-x}\,\mathrm{d}x$, for integer x, $\Gamma(x) = (x-1)!$. Now define the entropy of $\hat{P}_i$ as $E(\hat{P}_j) = -\sum_{i=1}^{n} \hat{P}_j(i)\ln\hat{P}_j(i)$. Then we have $\prod_{j=1}^{m} B(\gamma_{1j},\ldots,\gamma_{nj})$ equals*

$$\exp\left(\sum_{j=1}^{m}\left(-\gamma_j E(\hat{P}_j) - \frac{1}{2}(n-1)\ln(\gamma_j) + \sum_{i=1}^{n}\ln(P_j(i)) + d\right)\right).$$

Here d is a constant such that $\frac{1}{2}\ln(2\pi)n - 1 \leq d \leq n - \frac{1}{2}\ln(2\pi)$, where $\ln(2\pi) \approx 0.92$.

V. Algorithm for uniform prior distribution

Another prior that has been studied previously is the uniform distribution over a polyhedron. This can model the situation when we think the opponent is playing uniformly within some region of a fixed strategy, such as a specific Nash equilibrium or a "population mean" strategy based on historical data. Prior work has used this model to generate a class of opponents who are more sophisticated than opponents who play uniformly at random over the entire space [11]). For example, in rock-paper-scissors, we may think the opponent is playing a strategy uniformly out of strategies that play each action with probability within [0.31,0.35], as opposed to completely random over [0,1].

Let $v_{i,j}$ denote the jth vertex for player i, where vertices correspond to mixed strategies. Let p^0 denote the prior distribution over vertices, where $p^0_{i,j}$ is the probability that player i plays the strategy corresponding to vertex $v_{i,j}$. Let V_i denote the number of vertices for player i. Algorithm 2 computes the Bayesian best response in this setting. Correctness follows straightforwardly by applying Corollary 1 with the formula for the mean of the uniform distribution.

Algorithm 2 Algorithm for opponent exploitation with uniform prior distribution over polyhedron

Inputs: Prior distribution over vertices p^0, response functions r_t for $0 \leq t \leq T$

$M_0 \leftarrow$ strategy profile assuming opponent i plays each vertex $v_{i,j}$ with probability $p^0_{i,j} = \frac{1}{V_i}$
$R_0 \leftarrow r_0(M_0)$
Play according to R_0
for $t = 1$ to T **do**
  for $i = 1$ to N **do**
    $a_i \leftarrow$ action taken by player i at time step t
    for $j = 1$ to V_i **do**
      $p^t_{i,j} \leftarrow p^{t-1}_{i,j} \cdot v_{i,j}(a_i)$
    Normalize the $p^t_{i,j}$'s so they sum to 1
  $M_t \leftarrow$ strategy profile assuming opponent i plays each vertex $v_{i,j}$ with probability $p^t_{i,j}$
  $R_t \leftarrow r_t(M_t)$
  Play according to R_t

VI. Experiments

We ran experiments on the game described in Section IV-A. For the beta function computations we used the Colt Java math library.For our first set of experiments we tested our basic algorithm which assumes that we observe a single opponent action (Equation 1). We varied the Dirichlet prior parameters to be uniform in $\{1,n\}$ to explore the runtime as a function of the size of the prior (since computing larger values of the Beta function can be challenging). The results (Table I) show that the computation is very fast even for large n, with running time under 8 microseconds for $n = 500$. However, we also observe frequent numerical instability for large n. The second row shows the percentage of the trials for which the algorithm produced a result of "NaN" (which typically results from dividing zero by zero). This jumps from 0% for $n = 50$ to 8.8% for $n = 100$ to 86.9% for $n = 200$. This is due to instability of algorithms for computing the beta function. We used the best publicly available beta function solver, but perhaps there could be a different solver that leads to better performance in our setting (e.g., it trades off runtime for additional precision). Despite the cases of instability, the results indicate that the algorithm runs extremely fast for hundreds of prior observations, and since it is exact, it is the best algorithm for the settings in which it produces a valid output. Note that $n = 100$ corresponds to 400 prior observations on average since there are four parameters, and that the experiments in previous work used a horizon of 200 hands per match against an opponent [7].

We tested our generalized algorithm for different numbers of observations, using a fixed Dirichlet prior with all parameters equal to 2 as in prior work [7]. We observe (Table II) that the algorithm runs quickly for large numbers of observations, though again it runs into numerical instability for large values. As one example, it takes 19ms for $\theta_b = 101$, $\theta_s = 100$.

n	10	20	50	100	200	500
Time	0.0005	0.0008	0.0018	0.0025	0.0034	0.0076
NaN	0	0	0	0.0883	0.8694	0.9966

TABLE I
RESULTS OF MODIFYING DIRICHLET PARAMETERS TO BE U{1,N} OVER ONE MILLION SAMPLES. FIRST ROW IS AVERAGE RUNTIME IN MILLISECONDS. SECOND ROW IS PERCENTAGE OF THE TRIALS THAT OUTPUT "NAN."

n	10	20	50	100	200	500	1000
Time	0.015	0.03	0.36	2.101	10.306	128.165	728.383
NaN	0	0	0	0	0.290	0.880	0.971

TABLE II
RESULTS USING DIRICHLET PRIOR WITH ALL PARAMETERS EQUAL TO 2 AND θ_b, θ_s IN U{1,N} AVERAGED OVER 1,000 SAMPLES. FIRST ROW IS AVERAGE RUNTIME (MS), SECOND ROW IS % OF TRIALS PRODUCING "NAN."

We compared our algorithm against the three heuristics described in previous work [7]. The first heuristic Bayesian Best Response (BBR) approximates the opponent's strategy by sampling strategies according to the prior and computing the mean of the posterior over these samples, then best-responding to this mean strategy; Max A Posteriori Response heuristic (MAP) samples strategies from the prior, computes the posterior value for these strategies, and plays a best response to the one with highest posterior value; Thompson's Response samples strategies from the prior, computes the posterior values, then samples one strategy for the opponent from these posteriors and plays a best response to it. For all approaches we used a Dirichlet prior with the standard values of 2 for all parameters. For all the sampling approaches we sampled 1,000 strategies from the prior for each opponent and used these strategies for all hands against that opponent (as was done in prior work [7]). Note that one can draw samples x_i from a Dirichlet distribution by first drawing independent samples y_i from Gamma distributions each with density $\text{Gamma}(\alpha_i, 1) = \frac{y_i^{\alpha_i - 1} e^{-y_i}}{\Gamma(\alpha_i)}$ and then setting $x_i = \frac{y_i}{\sum_j y_j}$.We also tested a best response strategy that knows the actual mixed strategy of the opponent, not just a distribution over his strategies, as well as the Nash equilibrium strategy.[5] Note that the game has a value to us of -0.75, so negative values are not necessarily indicative of "losing."

Table III shows that our exact Bayesian best response algorithm (EBBR) outperforms the heuristic approaches, as expected since it is optimal when the opponent's strategy is drawn from the prior (though performance is very similar to BBR and not statistically distinguishable until 25 iterations). BBR performed best out of the sampling approaches, which is not surprising because it is trying to approximate the optimal approach while the others are optimizing a different objective. All of the sampling approaches outperformed just following the Nash equilibrium, and as expected all exploitation approaches performed worse than playing a best response to the opponent's actual strategy. Note that, against an opponent drawn from a Dirichlet distribution with all parameters equal to 2 and no further observations of his play, our best response would be to always call, which gives us expected payoff of zero. Thus for the initial column the actual value for EBBR when averaged over all opponents would be zero. Against this distribution the Nash equilibrium has expected payoff -0.375.

[5]Note that the Nash equilibrium for player 2 is to call a big bet with probability $\frac{1}{4}$ and a small bet with probability 1 (the equilibrium for player 1 is to always bet big with K and to bet big with probability $\frac{5}{6}$ with J).

Algorithm	Initial	10	25
EBBR	$\mathbf{-0.00003 \pm 0.0003}$	$\mathbf{-0.0004 \pm 0.0009}$	$\mathbf{0.0002 \pm 0.0008}$
BBR	-0.00003 ± 0.0003	-0.0004 ± 0.0009	-0.0065 ± 0.0008
MAP	-0.1649 ± 0.0002	-0.2025 ± 0.0007	-0.2664 ± 0.0007
Thompson	-0.2098 ± 0.0002	-0.2224 ± 0.0007	-0.2996 ± 0.0007
FullBR	0.4975 ± 0.0002	0.4971 ± 0.0006	0.4978 ± 0.0005
Nash	-0.3750 ± 0.0000	-0.3749 ± 0.0001	-0.3751 ± 0.0001

TABLE III
COMPARISON WITH ALGORITHMS FROM PRIOR WORK, FULL BEST RESPONSE, AND NASH EQUILIBRIUM USING DIRICHLET PRIOR WITH PARAMETERS EQUAL TO 2. SAMPLING ALGORITHMS USE 1000 SAMPLES. FOR INITIAL COLUMN WE SAMPLED 100 MILLION OPPONENTS FROM THE PRIOR, FOR 10 ROUNDS WE SAMPLED ONE MILLION, AND FOR 25 ROUNDS 500,000. RESULTS ARE AVERAGE WINRATE PER HAND OVER ALL OPPONENTS WITH 95% CONFIDENCE INTERVALS.

On the positive side the exploitation approaches (particularly EBBR and BBR) are able to significantly outperform the Nash equilibrium strategy when given access to a reliable prior distribution; however, none of them are able to improve over time as a result of additional observations (EBBR and BBR perform around the same with more observations while Thompson and MAP perform noticeably worse). This indicates that, for this setting at least, just observing the opponent's public action and not private information is not additionally useful in comparison to the performance variance and the noise introduced from sampling. In order to successfully learn beyond the prior in imperfect-information settings, algorithms will need access to some of the opponents' private information. Previous experiments had also shown that when the sampling approaches are played against opponents drawn from the prior, the winning rates converge, typically very quickly (even with access to the opponent's private information in certain hands that went to *showdown*): "The independent Dirichlet prior is very broad, admitting a wide variety of opponents. It is encouraging that the Bayesian approach is able to exploit even this weak information to achieve a better result." [7]

We also tested the effect of using only 10 samples of the opponent's strategy for the sampling approaches. The approaches would then have a noisier estimate of the opponent's strategy and should achieve lower performance against the actual strategy of the opponent, though run significantly faster.

Thompson and MAP performed very similarly using 10 vs. 1000 samples (these approaches essentially end up selecting a single strategy from the set of samples to be used as the model, and the results indicate that they are relatively insensitive to the number of samples used), but BBR performs significantly worse. While the performance between EBBR and BBR was statistically indistinguishable for 1000 samples, EBBR significantly outperforms BBR with 10 samples, particularly for more iterations. As before the sampling approaches seem to actually perform worse over time as the noise propagates,

Alg	Initial	10	25	100
EBBR	**.0001 ± .0003**	**-.0003 ± .0003**	**.0002 ± .0002**	**-.0014 ± .0005**
BBR	-.0662 ± .0003	-.0902 ± .0003	-.1634 ± .0002	-.3127 ± .0004
MAP	-.1699 ± .0002	-.2060 ± .0002	-.2657 ± .0001	-.3082 ± .0004
Thomp.	-.2118 ± .0002	-.2247 ± .0002	-.2844 ± .0001	-.3725 ± .0004
FullBR	.4976 ± .0002	.4973 ± .0002	.4975 ± .0001	.4969 ± .0003
Nash	-.3750 ± .0000	-.3750 ± .0000	-.3750 ± .0000	-.3750 ± .0001

TABLE IV

COMPARISON OF OUR ALGORITHM WITH ALGORITHMS FROM PRIOR WORK (BBR, MAP, THOMPSON), FULL BEST RESPONSE, AND NASH EQUILIBRIUM USING DIRICHLET PRIOR WITH PARAMETERS EQUAL TO 2. THE SAMPLING ALGORITHMS EACH USE 10 SAMPLES FROM THE OPPONENT'S STRATEGY (AS OPPOSED TO 1000 SAMPLES FROM OUR EARLIER ANALYSIS). FOR THE INITIAL COLUMN WE SAMPLED 100 MILLION OPPONENTS FROM THE PRIOR, FOR 10 AND 25 ROUNDS WE SAMPLED TEN MILLION, AND 300,000 FOR 100 ROUNDS.

while the performance of EBBR remains about the same. The dropoff of BBR is particularly significant. The results indicate that EBBR would be particularly preferable over the sampling approaches if the number of available samples is small (e.g., due to running time considerations) and as the number of game iterations increases (though eventually EBBR can run into numerical stability issues described earlier).

VII. CONCLUSION

One of the most fundamental problems in game theory is learning to play optimally against opponents who may make mistakes. We presented the first exact algorithm for performing exploitation in imperfect-information games in the Bayesian setting using the most well-studied prior distribution for this problem, the Dirichlet distribution. Previously an exact algorithm had only been presented for normal-form games, and the best previous algorithm was a heuristic with no guarantees. We demonstrated experimentally that our algorithm can be practical and that it outperforms the best prior approaches, though it can run into numerical stability issues for large numbers of observations.

We presented a general meta-algorithm and new theoretical framework for studying opponent exploitation. Future work can extend our analysis to many important settings. For example, we would like to study the setting when the opponent's private information is only sometimes observed (we expect our approach can be extended easily to this setting) and general sequential games where the agents can take multiple actions (which we expect to be hard, as indicated by the analysis in the tech report). We would also like to extend analysis for any number of agents. Our algorithm is not specialized for two-player zero-sum games (it applies to general-sum games); if we are able to compute the mean of the posterior strategy against multiple opponent agents, then best responding to this strategy profile is just a single agent optimization and can be done in time linear in the size of the game regardless of the number of opponents. While the Dirichlet is the most natural prior for this problem, we would also like to study other important distributions. We presented an algorithm for the uniform prior distribution over a polyhedron, which could model the situation where we think the opponent is playing a strategy from a uniform distribution in a region around a particular strategy, such as a specific equilibrium or a "population mean" based on historical data.

Opponent exploitation is a fundamental problem, and our algorithm and extensions could be applicable to many domains that are modeled as an imperfect-information games. For example, many security game models have imperfect information, e.g., [14], [15], and opponent exploitation in security games has been a very active area of study, e.g., [16], [17]. It has also been proposed recently that opponent exploitation can be important in medical treatment [18].

REFERENCES

[1] X. Chen and X. Deng, "Settling the complexity of 2-player Nash equilibrium," in *Proceedings of the Annual Symposium on Foundations of Computer Science (FOCS)*, 2006.

[2] R. Gibson, "Regret minimization in games and the development of champion multiplayer computer poker-playing agents," Ph.D. dissertation, University of Alberta, 2014.

[3] D. Koller, N. Megiddo, and B. von Stengel, "Fast algorithms for finding randomized strategies in game trees," in *Proceedings of the 26th ACM Symposium on Theory of Computing (STOC)*, 1994, pp. 750–760.

[4] S. Ganzfried and T. Sandholm, "Game theory-based opponent modeling in large imperfect-information games," in *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 2011.

[5] D. Fudenberg and D. Levine, *The Theory of Learning in Games*. MIT Press, 1998.

[6] G. W. Brown, "Iterative solutions of games by fictitious play," in *Activity Analysis of Production and Allocation*, T. C. Koopmans, Ed. John Wiley & Sons, 1951, pp. 374–376.

[7] F. Southey, M. Bowling, B. Larson, C. Piccione, N. Burch, D. Billings, and C. Rayner, "Bayes' bluff: Opponent modelling in poker," in *Proceedings of the 21st Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, July 2005, pp. 550–558.

[8] M. Johanson, K. Waugh, M. Bowling, and M. Zinkevich, "Accelerating best response calculation in large extensive games," in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2011.

[9] M. Johanson, M. Zinkevich, and M. Bowling, "Computing robust counter-strategies," in *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS)*, 2007, pp. 1128–1135.

[10] M. Johanson and M. Bowling, "Data biased robust counter strategies," in *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2009.

[11] S. Ganzfried and T. Sandholm, "Safe opponent exploitation," *ACM Transactions on Economics and Computation (TEAC)*, 2015, special issue on selected papers from EC-12.

[12] P. Tang, Z. Wang, and X. Zhang, "Optimal commitments in auctions with incomplete information," in *Proceedings of the ACM Conference on Economics and Computation (EC)*, 2016.

[13] S. Ganzfried and Q. Sun, "Bayesian opponent exploitation in imperfect-information games," *CoRR*, vol. abs/1603.03491, 2016. [Online]. Available: http://arxiv.org/abs/1603.03491

[14] J. Letchford and V. Conitzer, "Computing optimal strategies to commit to in extensive-form games," in *Proceedings of the ACM Conference on Electronic Commerce (EC)*, 2010.

[15] C. Kiekintveld, M. Tambe, and J. Marecki, "Robust Bayesian methods for Stackelberg security games (extended abstract)," in *Autonomous Agents and Multi-Agent Systems*, 2010.

[16] J. Pita, M. Jain, M. Tambe, F. Ordóñez, and S. Kraus, "Robust solutions to Stackelberg games: Addressing bounded rationality and limited observations in human cognition," *Artificial Intelligence Journal*, vol. 174, no. 15, pp. 1142–1171, 2010.

[17] T. H. Nguyen, R. Yang, A. Azaria, S. Kraus, and M. Tambe, "Analyzing the effectiveness of adversary modeling in security games," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2013.

[18] T. Sandholm, "Steering evolution strategically: Computational game theory and opponent exploitation for treatment planning, drug design, and synthetic biology," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2015, senior Member Blue Skies Track.

Geometry and Generation of a New Graph Planarity Game

Rutger Kraaijer
Dept. of Information and Computing Sciences
Utrecht University
r.kraaijer@gmail.com

Marc van Kreveld
Dept. of Information and Computing Sciences
Utrecht University
m.j.vankreveld@uu.nl

Wouter Meulemans
Dept. of Mathematics and Computer Science
Eindhoven University of Technology
w.meulemans@tue.nl

André van Renssen
School of Information Technologies
University of Sydney
andre.vanrenssen@sydney.edu.au

***Abstract*—We introduce a new abstract graph game, SWAP PLANARITY, where the goal is to reach a state without edge intersections and a move consists of swapping the locations of two vertices connected by an edge. We analyze this puzzle game using concepts from graph theory and graph drawing, computational geometry, and complexity. Furthermore, we specify what good levels look like and we show how they can be generated. We also report on experiments that show how well the generation works.**

***Index Terms*—planarity, graphs, puzzle games, puzzle complexity, instance generation**

I. INTRODUCTION

PLANARITY [15] is a popular abstract puzzle game that is widely available. Besides being a smartphone app and having a Wikipedia page, it is also available as a "model" in Netlogo [16]. The idea is that a tangled graph is given with intersecting edges, and the objective is to untangle the graph by dragging vertices to other locations. If the graph is planar (meaning that it can be embedded in the plane without intersections), then the objective can always be realized, and we never need more vertex drags than there are vertices.

Algorithmically, planarity of a graph can be tested in linear time [6], [8], [14], and the algorithm returns an embedding of the graph in which it is drawn planar. So for an algorithm, an instance of PLANARITY is easily solvable in linear time. Minimizing the number of moves, however, is NP-hard [9], [18], see also [4].

In this paper we propose several variations on the game PLANARITY. These variations essentially limit the freedom of the operations that can be done on the drawn graph. We will investigate one of the new variations closely: SWAP PLANARITY, where we can *swap the locations of two vertices* that are connected by an edge. Examples are shown in Fig. 1. We show that quadratically many swaps are sometimes necessary, even if the input has just one edge crossing, and we show that quadratically many swaps are always sufficient. We also show that the solvability question is NP-complete for general graphs. Simple graphs like trees can always be made planar by swaps, but we show that minimizing the number of swaps needed is NP-hard.

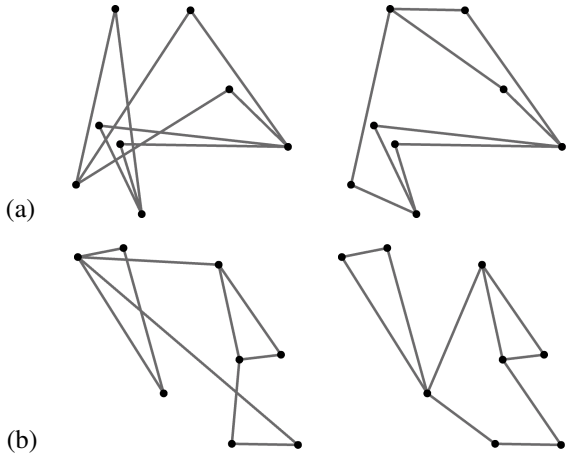


Fig. 1. (a) Puzzle and solution after one swap (the left, nearly vertical edge). (b) Puzzle and solution after two swaps.

We also investigate the automated generation of good puzzle instances. We describe a five-step process which yields a puzzle instance. Some of the considerations of a good instance are puzzle (complexity) based and some are geometry based. Our process guarantees that the puzzle and geometry criteria are met.

We implemented the generator and ran a number of experiments that uncover some properties of point set generation and puzzle diversity. The implementation includes a puzzle mode where the user can solve generated instances by hand.

II. GRAPH UNTANGLING PUZZLES

We will limit the operations that change the drawing of the graph to arrive at different puzzles. Since the puzzle type is abstract, it is necessary that the interaction and operations themselves are simple. The puzzle then becomes an elegant abstract puzzle of which there are many already (Move, Lines/Flow, Zengrams, Nintaii, Fling, and several more).

978-1-5386-4359-4/18/$31.00 ©2018 IEEE

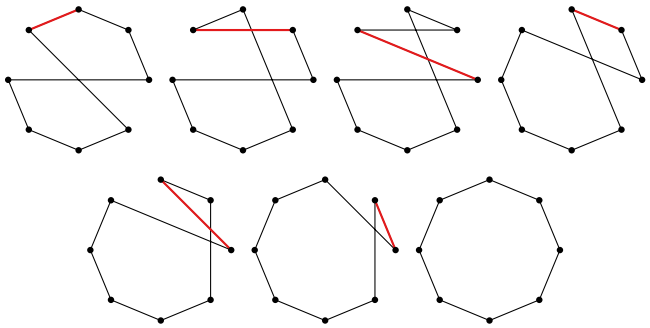


Fig. 2. Six steps to solve an 8-cycle with one intersection. The edge to be reversed is indicated.

Besides interacting with a vertex like in PLANARITY, it is natural to interact with an edge. Clicking or selecting is arguably the easiest interaction. We list a number of ways in which the graph drawing can change when an edge is selected:

- Swap: the two endpoints of the selected edge swap locations. Intuitively, the edge turns around while the endpoints drag all incident edges with them.
- Rotate: like swap, but now the selected edge rotates over 90 degrees around its center. Since a single edge can be selected consecutively three times, it does not matter whether we rotate clockwise or counter-clockwise.
- Stretch: the selected edge is scaled by a factor 2 from its center, or by a factor $1/2$.
- Collapse: the endpoints of the selected edge are united. The united vertex is placed in the middle of the edge and gets all edges incident to the original vertices. The selected edge is removed.

Of these versions, the first one distinguishes itself from the others because no new vertex positions appear. The graph will always be drawn on the original positions. Furthermore, the last version distinguishes itself by the fact that the number of vertices is reduced. Eventually, the whole graph could be reduced to a single vertex, so the challenge must be to remove all intersections in a limited number of steps. In the first three versions, steps are reversible.

We can also stay closer to the original PLANARITY puzzle and drag vertices in more limited ways. For example, a set of points can be given along with the graph, and the vertices must be dragged to the given points. This version is related to a well-known problem in the graph drawing research area, namely that of embedding a graph on a given set of points [5]. In essence, the initial drawing of the graph is irrelevant.

In this paper we concentrate on the swap version, named SWAP PLANARITY. It is perhaps the most elegant version and the graphs appearing after operations can be controlled in their appearance, unlike with the other versions (where edges may get so short that they cannot be selected any more). All following results concern this version.

Before we go into the algorithmic complexity of solving such puzzles and the process of generating good puzzle instances, we give a few examples to understand the puzzle better. First, consider the puzzle instance in Fig. 2 with eight vertices and eight edges. The graph is a single cycle and it has only one intersection. To solve this puzzle, note that any swap will *increase* the number of intersections. The minimum number of swaps needed is six; the set of intermediate drawings is shown in the figure and the selected edge is shown. When we extend this example to a set of n vertices and edges, then we need $\Omega(n^2)$ swaps to solve the instance.

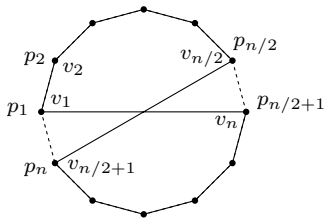


Fig. 3. Positions and vertices for the lower bound construction.

Lemma 1: There exist graphs with n vertices that require $\Omega(n^2)$ swaps to obtain a plane drawing.

Proof. Consider the drawing of Fig. 2 generalized to n vertices, with n even. Name the vertices of the graph $v_1, \ldots, v_n$ so that $v_1, \ldots, v_{n/2}$ are clockwise and $v_{n/2+1}, \ldots, v_n$ are counter-clockwise. This implies that the edges (v_1, v_n) and $(v_{n/2}, v_{n/2+1})$ intersect. Let us name the positions for the vertices $p_1, \ldots, p_n$, where initially v_1 is at p_1 and the positions are numbered clockwise, see Fig. 3.

In total there are $2n$ ways to place $v_1, \ldots, v_n$ on $p_1, \ldots, p_n$ without intersections: in cyclic order clockwise or counter-clockwise, and starting anywhere. This means that either $v_1, \ldots, v_{n/2}$ or $v_{n/2+1}, \ldots, v_n$ must be reversed on the positions $p_1, \ldots, p_n$.

Listing the node identifiers in the order of the cycle $v_1, \ldots, v_n$, we initially get the cyclic sequence $p_1, \ldots, p_{n/2}, p_n, \ldots, p_{n/2+1}$. A swap exchanges precisely two adjacent elements (where the first and last are also adjacent). Thus, to sort this sequence in one of the $2n$ ways, at least $\binom{n/2}{2} = \Omega(n^2)$ swaps are needed. □

The next lemma shows that quadratically many swaps are sufficient; the result has been proved before as node swapping [19].

Lemma 2: Every graph with n vertices that has a solution has a sequence of $O(n^2)$ swaps to obtain a plane drawing.

Proof. Assume first that the graph has a single connected component. Name the positions $p_1, \ldots, p_n$, and name the vertices of the graph $v_1, \ldots, v_n$ in such a way that the graph is drawn plane if v_i is at position p_i. We prove by induction that any connected graph with n vertices can place its vertices at $v_1, \ldots, v_n$ at positions $p_1, \ldots, p_n$, respectively.

Choose any vertex v_j such that its removal will leave the graph connected. Suppose a vertex v_k is currently at position p_j. Use the path between v_j and v_k in G to get v_j onto p_j as follows. Suppose this path is $v_k = w_1, w_2, \ldots, w_h = v_j$,

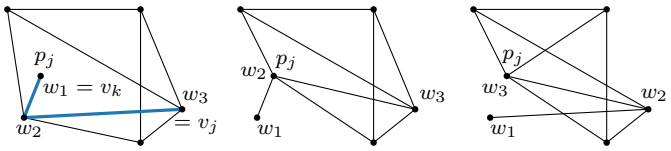


Fig. 4. Bringing v_j to p_j using the bold path w_1, w_2, w_3 (note that all plane embeddings must have v_j at p_j). Left: initial situation. Middle: after swap (w_1, w_3). Right: after swap (w_2, w_3).

see Fig. 4 for an example. We swap (w_1, w_2), then (w_2, w_3), and so on until (w_{h-1}, w_h). This brings v_j onto p_j in $h - 1 = O(n)$ swaps. We remove v_j from the graph and p_j from the locations and continue inductively. It is clear that at most $O(n^2)$ swaps are needed in total. If the graph has multiple connected components, we follow this procedure for each connected component. □

Another puzzle variant of swapping to planarity is possible, namely where we swap any two vertices (so they need not be connected by an edge). The interaction with the puzzle consists of clicking on two different vertices consecutively. In this variation, any solvable puzzle instance with n vertices is solvable in at most $n - 1$ swaps, because we can directly bring any vertex to the correct position. The challenge of this variant reduces to recognizing where vertices need to be to get a planar embedding, and no longer how to get it there.

III. Complexity of Swap Planarity

Theorem 3: Given an embedded graph G, it is NP-complete to decide if the graph can be made planar using swaps.

Proof. A solution of the problem can be presented by the sequence in which vertices are swapped. This solution can be represented in $O(n^2)$ space by Lemma 2. Swapping these vertices and checking if the resulting graph is plane can be done in polynomial time, hence the problem is in NP.

Cabello [5] showed that it is NP-complete to decide if a given point set P admits a planar drawing of a given graph G where the vertices must be placed at the points. This is also true for connected graphs. Given an instance of this problem with a connected graph, we assign the vertices of G to the points in P arbitrarily.

We now solve the graph planarization using swaps on this embedding of the graph. If it has a solution, we can just output the final point-vertex relation, leading to a planar embedding of the given graph. If no solution exists, we also know that no planar embedding exists, since by the proof of Lemma 2, we can realize any assignment of vertices to points in a connected graph. □

It is known that if G is a tree, the embedding problem of G onto P is no longer NP-complete because every tree can be embedded without intersections onto a planar point set [3], [13]. This does not imply that our puzzle game is easy to solve when the graph is a tree *when we bound the number of swaps*. In particular, we can show that deciding whether the vertices of an embedded tree can be swapped to become plane in at most k swaps is NP-complete.

Theorem 4: Given an embedded tree on n vertices, it is NP-complete to decide if k swaps suffice to obtain a plane drawing.

Proof. **(sketch)** We reduce from positive planar 1-in-3-SAT, which was shown to be NP-complete by Mulzer and Rote [11].

Positive planar 1-in-3-SAT. In the positive planar 1-in-3-SAT problem we are given a collection of clauses, each consisting of exactly three variables. Each of these variables occurs positively in the clause. In addition, we are given a planar embedding of the clauses and variables such that a variable is connected to a clause if and only if the variable occurs in the clause. The positive planar 1-in-3-SAT problem is to decide if there exists a truth assignment to the variables such that for each clause exactly one variable is true.

Variable gadget. We construct a variable gadget as follows by placing two rows of vertices with a row of vertices vertically between them and adding edges as shown in Fig. 5. In order to remove the created crossings using the minimum number of swaps, we need to swap the endpoints of two of the four vertical edges (first and third or second and fourth).

Split gadget. In order to connect the variable to clauses, we construct a split gadget by taking the variable gadget and adding an extension to one of the vertices of a column (see Fig. 6a). There are two minimal ways of removing the crossings, each costing two swaps. The first is by swapping (v_6, v_7) and (v_{13}, v_{14}) (see Fig. 6b). Note that (v_{13}, v_{14}) is forced to be swapped, since otherwise (v_6, v_{10}) crosses (v_7, v_{14}). The other option is to swap (v_1, v_2) and (v_{11}, v_{12}) (see Fig. 6c). Note that in this case, (v_{13}, v_{14}) can be swapped, but doing so implies that we use more than the minimal number of swaps to make the tree plane.

Clause gadget. The construction of a clause gadget is shown in Fig. 7. We place a central vertex v_1 and we place

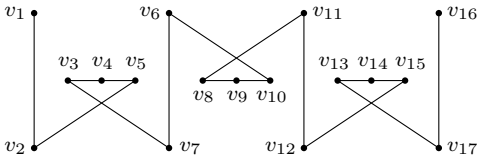


Fig. 5. The variable gadget.

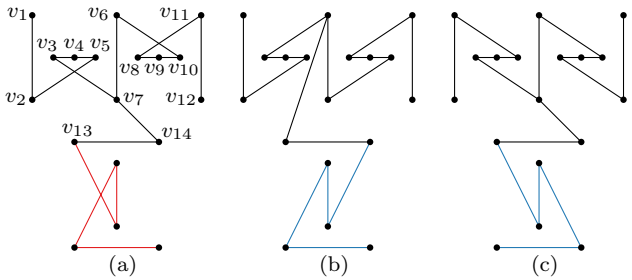


Fig. 6. (a) The split gadget and its two minimal planarizations: (b) the TRUE assignment and (c) the FALSE assignment.

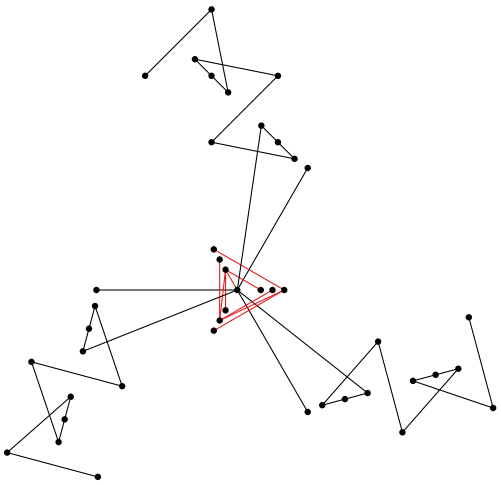


Fig. 7. The clause gadget.

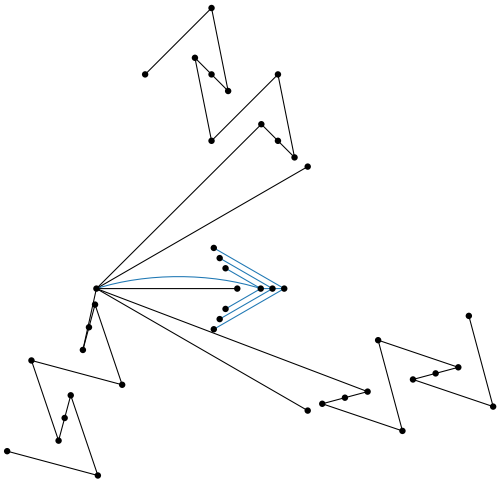


Fig. 8. A clause satisfied by the bottom-left variable.

three vertices v_2, v_3, and v_4 equidistant from it, connecting them to v_1. Next, we place three layers of three vertices each equidistant from v_1 such that each layer forms a triangle containing the central vertex. We note that the only way to untangle this structure is to swap locations of the central vertex with one of v_2, v_3, and v_4 and orient the three layers in such a way that the edges missing in each layer line up towards the new location of v_1. This takes three swaps.

We connect a variable to a clause by first adding a split gadget and connecting the path that was split off to the central vertex of the clause. We connect each variable to the clause using two crossing steps, similar to the variable gadget. In the full version of this proof, we argue that a clause gadget can be untangled using five swaps if and only if exactly one of its variable gadgets satisfies the clause. We also argue that there is no globally different solution that uses this number of swaps, yet is not a valid solution to the 1-in-3-SAT instance.

Constructing a tree. We note that the graph described above is not necessarily a tree as it can contain cycles. To construct a tree, we remove the middle edges from some of the chains (see Fig. 9). Since the endpoints of these edges are never swapped in any satisfiable assignment, this does not influence the satisfiability of the instance.

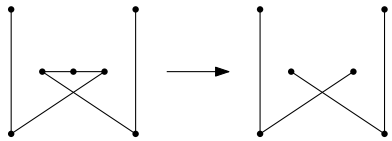


Fig. 9. Removing edges to construct a tree.

This last step shows that we can solve an instance of positive planar 1-in-3-SAT by constructing a tree and determining whether the clause gadgets can be untangled using five swaps per clause. Retrieving the variable assignment for positive planar 1-in-3-SAT can be done by checking how the corresponding variable gadgets are untangled. Hence, the problem is NP-complete. □

IV. Generating levels

In this section we describe how puzzle instances or levels can be generated for Swap Planarity. First we outline a five-step procedure, and then we explain these steps in more detail. We pay attention to three properties: (i) the puzzle instance should look good, also in states to be reached later, (ii) every possible good puzzle instance should be a possible output, for diversity, and (iii) solutions should not have a particular structure that might be identified by a puzzler, which may upset the intended puzzle instance difficulty.

A. Process of level generation

We describe a five-step procedure to generate a puzzle instance. We assume that a desired number n of vertices is specified, and also a desired number m of edges, and a desired minimal number s of swaps to the solution.

1) Generate a set V of n points in a playing area, such that for no two points, an edge between them would visually conflict with any other point from V (property (i)).
2) Generate a Delaunay triangulation on V, leading to an edge set E''.
3) Perform a number of Lawson flips to make sure that the solution of the puzzle instance need not only have Delaunay edges (done for properties (ii) and (iii)). This makes E' out of E''.
4) Remove a number of edges at random from E' until m edges remain. Make sure that no isolated vertices remain. This gives the edge set E.
5) Perform s swap operations at random, by picking edges at random from E. Test if the resulting instance requires s swaps to a planar state (and if not, swap more edges).

The whole process ensures property (ii): any puzzle instance that satisfies property (i) can be generated, provided that sufficiently many flips are performed in step 3.

B. Generating points

Given the shape of screens, it is natural to generate a point set in a square or rectangular region. There are two important issues to consider when generating point sets. First, collinearity or near-collinearity of points means that potentially, an edge will partly overlap a vertex in the drawing. This is undesirable. Second, point sets are "combinatorially different", which relates to the variation to be obtained in puzzle instances. We discuss these two issues next.

Let us assume that each vertex is drawn as a disc with radius r and every edge is drawn as a rectangle with length the distance between its endpoints and width $w < 2r$. Then any two vertices (centers) should be separated more than $2r$ in order for their discs to be disjoint. Furthermore, each vertex should be further away than $r + w/2$ from any edge it is not incident to [17]. To have a little more room around each vertex and edge we introduce a parameter δ that specifies for each point how far it must be from each other point and edge, when points are viewed as 0-dimensional and edges as 1-dimensional. We always choose $\delta > 2r$.

To generate a point set with these characteristics, we incrementally add points, uniformly distributed in a square. For each addition, we check if the vertex-vertex distance condition is violated or the collinearity is violated, and if so, we discard the last added point. We need to test only the new point against the previously accepted points. Hence, testing the vertex-vertex distance is easy in linear time per new point. To test collinearity, we choose the new point and every pair of accepted points. Using a bit of geometry we can identify a region bounded by four lines where the new point may not lie, see Fig. 10. The four lines are the inner and outer tangents to two discs of radius δ centered on the two accepted points. Hence, the test for collinearity can be done in quadratic time per new point.

When we generate puzzles with a considerable number of points we may get many failures. It is possible to compute the whole region where new points can be placed by generating the quadratically many regions for the accepted points and computing their union. The complement of this union is where a new point can still lie. In particular, we can compute this union and sample the complement explicitly, which means we do not get failures. If the union covers the whole square, we cannot add points anymore. The union itself, for p accepted points so far, has complexity $O(p^4)$ and can be computed in $O(p^4)$ time [10].

We next discuss the issue of combinatorially different point sets. To understand what this means, imagine a set of n points in convex position: they all lie on the convex hull. Whether points lie as the vertices of a regular n-gon, or spread on an ellipse, or more randomly placed (but still in convex position), these point sets are essentially the same from the perspective of intersecting edges between these points. If we have any graph on these points, the same edges will intersect and it does not matter where the points lie precisely. Moreover, any point set with n points of which $3 \leq k \leq n$ lie on the convex hull has at most $3n - k - 3$ edges that do not intersect (the fewer vertices on the convex hull, the more edges can be in a plane graph). Point sets with the same number of points but different numbers of points on the convex hull are combinatorially different. But there are still differences between point sets with the same numbers of points and the same number of points on the convex hull.

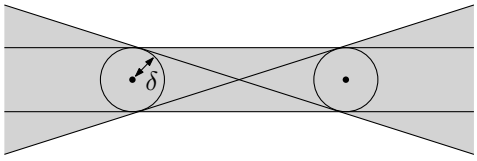


Fig. 10. Region where a third point may not be placed if collinearity should be avoided.

Two point sets V and W of n points each are *combinatorially the same* if and only if a one-to-one mapping f from one point set V to the other point set W exists such that for any three points $a, b, c \in V$, the sequence abc is a left turn if and only if the sequence $f(a)f(b)f(c)$ of points from W is a left turn. The equivalence class thus obtained is called an *order type* [1], [2]. Two point sets of the same order type allow exactly the same puzzle instances, and two point sets of different order types allow different sets of puzzle instances (but which might have some instances the same). So order types are directly related to the diversity of puzzle instances that can be made. In order to generate a variety of puzzles with, say, eight vertices, it will be useful to choose vertex locations with different order types in the point generation part. We will not discuss this further in this paper.

C. Generating a plane graph

Once we have generated a set V of n points without collinearity or closeness, we can generate edges. We generate a plane graph (a solution) to a puzzle instance in three steps (steps 2–4).

First, we compute the Delaunay triangulation of V [7]. This is a specific triangulation of a point set that maximizes the smallest angle that is used in the triangulation. If is also characterized by the empty-circle property: for any two points v_i and v_j for which a circle exists that touches only v_i and v_j and which has no points of V inside, there is an edge connecting v_i and v_j. This characterization (in general) completely specifies the triangulation. There are several known algorithms to compute the Delaunay triangulation of n points in $O(n \log n)$ time. This gives the edge set E''.

Second, we perform a few Lawson flips (beware that flips and swaps are very different operations). A Lawson flip can be applied to a pair of edge-adjacent triangles in a triangulation if those triangles together form a *convex* quadrilateral. A Lawson flip removes the shared edge and re-triangulates the resulting quadrilateral in the (only) other way. The reason to perform these flips is to make it harder for a puzzler to solve instances. Delaunay triangulations favor shorter edges, and Lawson flips can generate longer edges again. If a puzzler would know—or

realize—that the solution to each puzzle instance uses only Delaunay edges, then (s)he can quickly see which edges must be avoided in the drawing by imagining the empty-circle test (let's face it: these puzzles are going to be done by geometers). Edges to be flipped are selected randomly, and the flip is done only if the four involved vertices are in convex position (otherwise the resulting drawing would be non-planar). The resulting edge set is denoted E'.

Third, we remove some edges from E' so that a puzzle instance solution is not always a triangulation. We take care to not create any isolated vertices. These would not influence the puzzle or its solution in any way. Notice that an isolated edge does influence the puzzle. While a swap applied to such an edge does not change the drawing, swapping other edges may resolve edge intersections with the isolated edge.

By removing edges we can realize a desired number of edges in the solution. Removing many edges may cause the puzzle instance to have multiple solutions and become easy.

D. Generating an instance

We have now generated a graph with a specified number of vertices and edges, and in particular, a solution to this puzzle instance. To generate the puzzle instance itself we make some swaps such that undoing these (swapping the same edges in reverse order) solves the instance.

It appears that puzzle instances with just two or three swaps from a solution are already not so easy (Fig. 1). Once a player gets more experienced, instances with four swaps may become suitable. This means that testing the difficulty of a solution can be done by brute-force. For example, a graph with 20 edges that should be four swaps away from a solution can be tested by trying all $20 \cdot 19^3 = 137,180$ possibilities (we exclude swapping the same edge twice in a row).This may lead to an instance with fewer necessary swaps to solve than we have used to generate it; in this case we perform extra swaps until the desired minimal number of swaps is obtained. We will also recognize if there are more ways to a solved state, making the instance a bit easier too. Finally, swaps that are independent and possibly even well-separated also give rise to easier instances. Two swaps are independent if the four endpoints of the edges are disjoint and there is no other edge than the two that are swapped between these four vertices.

Observe that we have realized the three properties we aimed for. The visual quality (i) of the instance and every intermediate state that can be reached is captured by the vertex-vertex distance and vertex-edge distance conditions. The puzzle diversity (ii) is realized by allowing any number of vertices, edges, and steps to the solution, every possible plane drawing as a solution, and every possible non-plane drawing as a puzzle instance. There is no puzzle instance that cannot be generated. Absence of unintended structure (iii) is accomplished by ensuring that for a point set, any edge between two points could be part of the solution.

V. Implementation and Experiments

The Swap Planarity game is implemented using Unity. Besides trying the game to see how difficult and fun puzzle instances are, we are interested in the efficient generation of non-collinear point sets, the number of points on the convex hull, the non-collinearity parameter δ, and relations these.

Fig. 11 shows the interface. From the settings on the right we can see that the instance has 11 points generated with $\delta = 0.03$ to ensure non-collinearity, the initial triangulation is 3 flips away from being Delaunay, then 4 edges were removed and two swaps were performed to shuffle the planar graph. The solution is shown on the right.

When we try to generate a large point set with a large value of δ, we may fail because there may not be enough space on the screen (play area) to realize the separation. This also depends on the random generation itself. It can happen that a point set of 14 points cannot be extended to 15 points without violating collinearity, but sets of 15 non-collinear points may still exist. This means that the point generation procedure may have to abort and restart. If aborting is done too early, generation may be inefficient because we start from scratch without having to. If aborting is done too late, generation

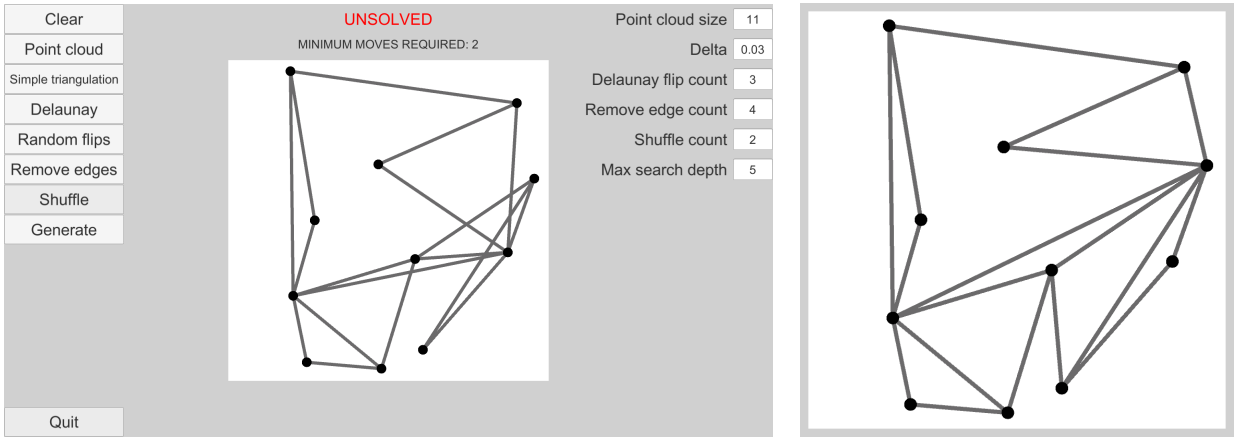


Fig. 11. Left figure, screenshot with the steps of the generation listed in sequence (Generate does all steps in order) and the settings used. Right figure, the solution of this puzzle instance.

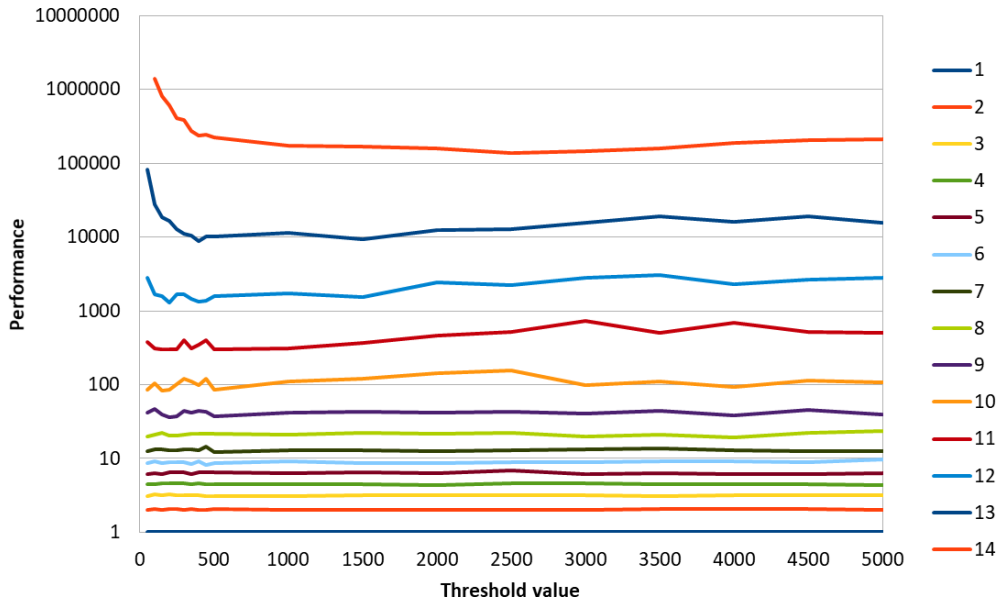


Fig. 12. Performance (total number of attempts to add a point to generate a complete point set) as a function of threshold choice, for different point set sizes. The threshold value represents the total number of attempts to add a random point before the generation of a point set is aborted and restarted.

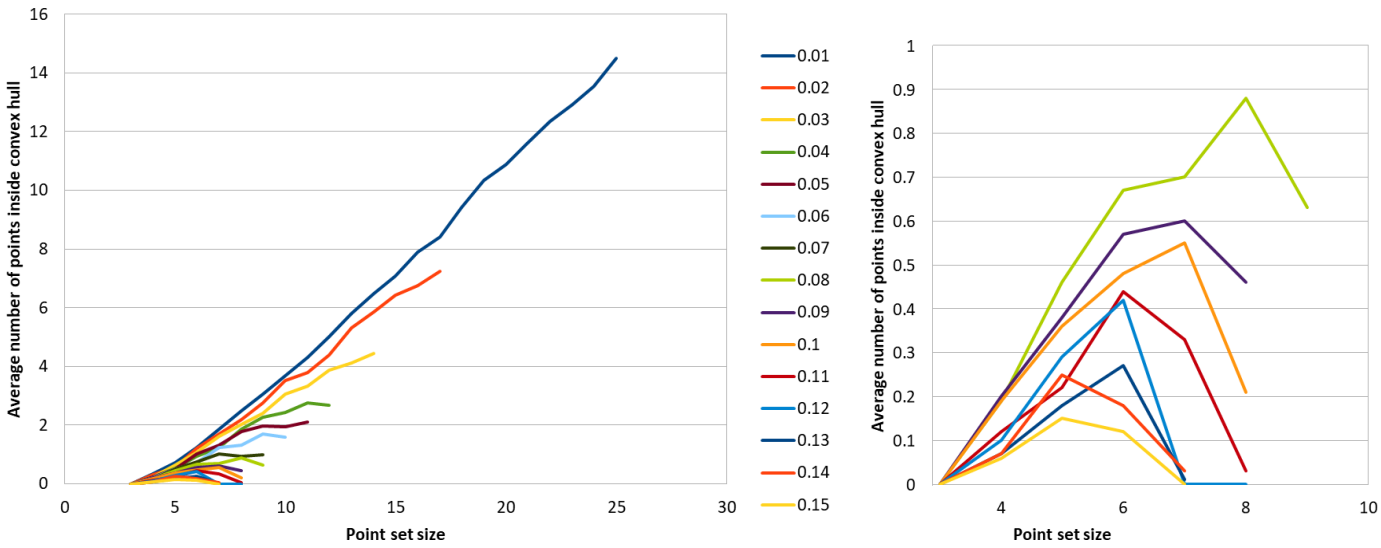


Fig. 13. Number of points inside the convex hull as a function of the point set size, for different thresholds. Data points are averaged over 100 point set instances that were generated. To the right, detail of the same figure.

may have spent a lot of time on a configuration that cannot be extended anymore. Fig.12 illustrates this for a fixed value of δ; data points we generated with intervals of 50 between 0 and 500 and with intervals of 500 after that. Note that the vertical axis has exponential scale. For the larger point set sizes we observe that we should make enough attempts to add a point, but not too many, to get the best efficiency.

We also determined the number of points inside the convex hull for different point set sizes and different values of δ. We noticed a surprising phenomenon: the larger δ, the fewer points are in the convex hull. This can be seen in Fig. 13, right: for increasing δ, fewer points tend to lie inside the convex hull. This happens especially when it gets difficult to generate point sets of a size with a δ, and hence we cannot observe the behavior for larger point set values in Fig. 13, left. It may be the case that a placement of points on the convex hull is a good placement if one wants to realize a large δ. This suggestion is supported by theory on bold graph drawings [12]. Fig. 14 shows the standard deviations over the 100 point set instances. It also shows that if δ is chosen relatively large, fewer points will be inside the convex hull.

The experiments show the following trade-off: puzzle instances with a good visual appearance (clear non-collinearity, large δ) are harder to generate efficiently and show less diversity, indicated by the relatively large number of points on the convex hull.

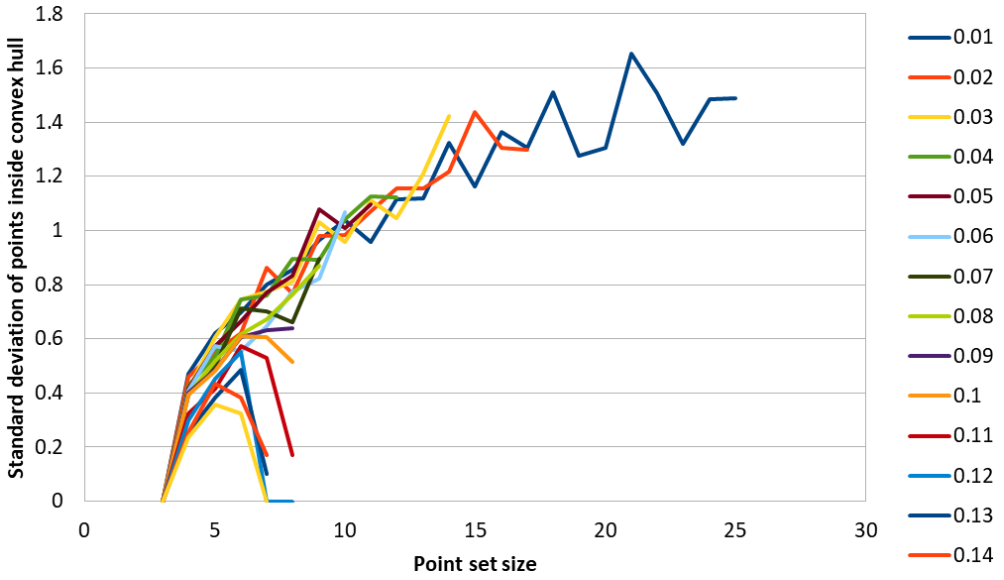


Fig. 14. Standard deviation of the number of points inside the convex hull as a function of the point set size, for different thresholds.

VI. Conclusions

We introduced a new graph planarity puzzle game called Swap Planarity and analyzed various properties, including the algorithmic complexity of solving instances. We presented a method to generate instances effectively while paying attention to visual clarity, diversity, and absence of accidental structure. Our implementation shows that generation works well, but has a trade-off between a good visual clarity on the one hand and diversity and efficient generation on the other.

We believe that the new, swap-based graph planarity puzzle game is a nice, elegant addition to the collection of abstract puzzle games. The puzzle is NP-hard, the number of crossings may need to be increased to reach a solution, and even small instances are not so easy to solve.

Acknowledgments

Part of this work was performed at the Dutch-Japanese Bilateral Seminar on Kinetic Geometric Networks. We thank the participants of the workshop for providing a fun and stimulating research environment. A.v.R. was supported by JST ERATO Grant Number JPMJER1201, Japan. M.v.K. was supported by the Netherlands Organisation for Scientific Research on grant no. 612.001.651. W.M. was supported by Netherlands eScience Center (NLeSC) on grant no. 027.015.G02.

References

[1] Oswin Aichholzer, Franz Aurenhammer, and Hannes Krasser. Enumerating order types for small point sets with applications. *Order*, 19(3):265–281, 2002.

[2] Oswin Aichholzer, Thomas Hackl, Clemens Huemer, Ferran Hurtado, Hannes Krasser, and Birgit Vogtenhuber. On the number of plane geometric graphs. *Graphs and Combinatorics*, 23(1):67–84, 2007.

[3] Prosenjit Bose. On embedding an outer-planar graph in a point set. *Computational Geometry*, 23(3):303–312, 2002.

[4] Prosenjit Bose, Vida Dujmovic, Ferran Hurtado, Stefan Langerman, Pat Morin, and David R. Wood. A polynomial bound for untangling geometric planar graphs. *Discrete & Computational Geometry*, 42(4):570–585, 2009.

[5] Sergio Cabello. Planar embeddability of the vertices of a graph using a fixed point set is NP-hard. *Journal of Graph Algorithms and Applications*, 10(2):353–363, 2006.

[6] Norishige Chiba, Takao Nishizeki, Shigenobu Abe, and Takao Ozawa. A linear algorithm for embedding planar graphs using PQ-trees. *Journal of Computer and System Sciences*, 30(1):54–76, 1985.

[7] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry – Algorithms and Aplications*. Springer, Berlin, 3rd edition, 2008.

[8] Hubert De Fraysseix, János Pach, and Richard Pollack. How to draw a planar graph on a grid. *Combinatorica*, 10(1):41–51, 1990.

[9] Xavier Goaoc, Jan Kratochvíl, Yoshio Okamoto, Chan-Su Shin, and Alexander Wolff. Moving vertices to make drawings plane. In *Graph Drawing, 15th International Symposium, GD 2007*, volume 4875 of *Lecture Notes in Computer Science*, pages 101–112. Springer, 2008.

[10] Dan Halperin and Micha Sharir. Arrangements. In Jacob E. Goodman, Joseph O'Rourke, and Csaba D. Tóth, editors, *Handbook of Discrete and Computational Geometry*, pages 723–762. 3rd edition, 2018.

[11] Wolfgang Mulzer and Günter Rote. Minimum-weight triangulation is NP-hard. *Journal of the ACM*, 55(2):11:1–11:29, 2008.

[12] János Pach. Every graph admits an unambiguous bold drawing. In *International Symposium on Graph Drawing*, pages 332–342, 2011.

[13] János Pach, Peter Gritzmann, Bojan Mohar, and Richard Pollack. Embedding a planar triangulation with vertices at specified points. *American Mathematical Monthly*, 98(DCG-ARTICLE-2008-010):165–166, 1991.

[14] Roberto Tamassia. *Handbook of Graph Drawing and Visualization*. CRC Press, 2013.

[15] John Tantalo. Planarity. http://planarity.net/, 2007. Accessed: 2018-05-25.

[16] Seth Tisue and Uri Wilensky. Netlogo: A simple environment for modeling complexity. In *International Conference on Complex Systems*, volume 21, pages 16–21, 2004.

[17] Marc van Kreveld. Bold graph drawings. *Computational Geometry*, 44(9):499–506, 2011.

[18] Oleg Verbitsky. On the obfuscation complexity of planar graphs. *Theoretical Compututer Science*, 396(1-3):294–300, 2008.

[19] Katsuhisa Yamanaka, Erik D. Demaine, Takehiro Ito, Jun Kawahara, Masashi Kiyomi, Yoshio Okamoto, Toshiki Saitoh, Akira Suzuki, Kei Uchizawa, and Takeaki Uno. Swapping labeled tokens on graphs. *Theoretical Computer Science*, 586:81–94, 2015.

Ensemble Decision Making in Real-Time Games

Philip Rodgers
Computer and Information Science
University of Strathclyde
Glasgow, United Kingdom
philip.rodgers@strath.ac.uk

John Levine
Computer and Information Science
University of Strathclyde
Glasgow, United Kingdom
john.levine@strath.ac.uk

Damien Anderson
Computer and Information Science
University of Strathclyde
Glasgow, United Kingdom
damien.anderson@strath.ac.uk

***Abstract*—This paper describes an Ensemble Agent for the classic arcade game Ms. Pac-Man. Our approach decomposes the problem into sub-goals. An expert agent is created for each sub-goal, with all experts reporting to a central arbiter. Our Ensemble Agent has achieved the AI world record for the arcade version of Ms. Pac-Man with a score of 162,280. For comparison, a MCTS-based monolithic agent was also created, based on the same accurate forward model that the Ensemble Agent uses, reaching a score of 115,180.**

***Index Terms*—ensemble, mcts, pac-man, real-time, decision**

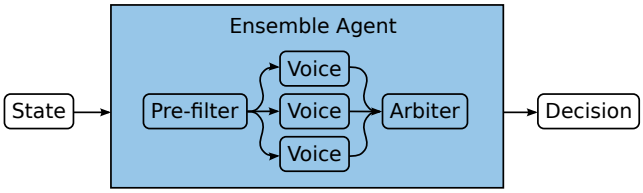


Fig. 1: Ensemble Decision System

I. Introduction

A. Ensemble Systems

Ensemble systems have been used for classification problems since the late 1970s [1]. They made use of feature partitioning to create multiple classifiers. These early classification systems used ensembles for a number of reasons.

- If there is too much data, divide and conquer techniques can be applied to split the data into more manageable subsets. Each subset can be run through a separate instance of the classifier.
- Too little data can be used more efficiently by creating multiple, overlapping training sets, with each set being used to train a separate classifier.
- A level of redundancy can be achieved by creating multiple classification systems, each trained on a unique or overlapping subset of the training data. The chances of misclassification can be lower for an ensemble classification system than for a monolithic system.
- Ensemble systems can easily handle heterogeneous data. A separate classifier can be built for each data source, and the ensemble system can combine the results for an overall classification.

A modern example of a powerful ensemble system is IBM's *Watson* [2]. Watson was originally created to play the TV quiz show *Jeopardy*, but has since been opened up for general use. It uses natural language recognition to analyse questions and generate queries. It sends the queries to multiple sources of answers, known as *many experts*, and combines the answers, calculating confidence levels for each of the answers. Watson has shown great potential in helping doctors with patient diagnoses [3].

The Ensemble decision system described in this paper uses concepts from ensemble classification systems, namely feature partitioning and expert systems, and applies them to real-time decision making. The Ensemble decision systems can be used to build complex agent behaviors out of simple components or *voices*. Each voice can be considered an agent in its own right, but with a simple, single goal or task. At the heart of the system is an *arbiter* which takes the outputs, or *opinions*, of the voices and generates the final decision, as shown in Figure 1.

The concept sounds similar to that of a *subsumption architecture*, but it differs significantly in that the decision making is not being deferred from one component to the next. Instead, all of the voices have an opinion all of the time, with each voice contributing to the net result. Ensemble decision systems far more akin to ensemble classification systems than subsumption architectures.

Ensemble decision systems are inherently flexible. They can have voices added or removed without affecting the other voices or the arbiter, only the final decision. Greater efficiency can also be achieved through the separation of behaviours that can be handled either reactively or deliberatively. Ideally all voices would be reactive, but some behaviours, such as ghost avoidance in Ms. Pac-Man, require some deliberation in order to achieve good results. These deliberative voices can be kept as simple as possible by disregarding anything unrelated to its given task. Pre-filtering is an option step that removes any moves known to be invalid or determined to be bad from previous iterations.

In its simplest form, the Ensemble Decision System was envisioned to have three primary component classes, short-, middle- and long-range goals. These can be seen as survival, tactics and strategy. This idea is by no means a requirement, and the Ms. Pac-Man ensemble agent described in this paper does not strictly adhere to this structure.

978-1-5386-4359-4/18/$31.00 ©2018 IEEE

B. Pac-Man

The original Pac-Man is an arcade machine from 1980, created by game designer Toru Iwatani of Namco and released in the United States by Midway Manufacturing Corporation. Pac-Man was a massive success, being the second highest grossing arcade game after emphSpace Invaders.

In Pac-Man, the player controls the main character around a maze using a four-way joystick. The aim of the game is to complete each level by eating all the pills dotted around the maze whist avoiding the four antagonistic ghosts. Each level also has four *power-pills* that allow Pac-Man to become *energised* for a short period of time, during which the ghosts can be eaten. Eating a power-pill also has the effect of making the ghosts reverse direction. In later levels the time that the ghosts are edible drops to zero. Two bonus items, often referred to as *fruit*, also appear per level.

Points are scored by eating normal pills (10 points each), power pills (50 points each), ghosts (200, 400, 800 and 1600 points if eaten in succession) and bonuses (100, 300, 500, 700, 1000, 2000, 3000 or 5000, depending on the bonus). From level 13 onwards, the bonus is always worth 5000 points.

C. Ms. Pac-Man

Ms. Pac-Man is a sequel to Pac-Man. It has essentially the same game mechanics but with several enhancements over the original game. There are four mazes, as opposed to the single maze of Pac-Man. The bonuses move around the maze, rather than appearing stationary in the centre of the maze, and in the later levels the bonus item is chosen pseudo-randomly.

The main difference, and what makes Ms. Pac-Man more appealing to players and AI developers alike, is the non-deterministic behaviour of the ghosts. The original Pac-Man game is entirely deterministic, so players can learn patterns to complete each level. The game can be beaten over and over by the simple repetition of the correct pattern. Ms. Pac-Man introduced enough random behaviour in the ghosts to allow for strategies but not patterns.

Ms. Pac-Man was chosen for this project as it is well known and there is a lot of prior AI research for this game [4]. Most of the prior work has been done using either the screen-capture Ms. Pac-Man competition framework[1], or the Ms. Pac-Man vs. Ghost-Team framework[2]. This project uses our own emulator, written in Java and capable of playing the original Ms. Pac-Man code. The emulator is described in section IV.

The original game has more complexity than the Ms. Pac-Man vs. Ghost-Team framework, and while the screen-capture framework is true to the original game, it has its own usability issues. The use of a emulator written in Java gives the simplicity of the Ms. Pac-Man vs. Ghost-Team framework and the authenticity of the screen-capture framework.

Ms. Pac-Man is a good benchmark of an AI system as it combines simplicity and difficulty. There are at most four possible options to choose from for any given state, and the search space is confined to a single-screen maze. Despite this simplicity, Ms. Pac-Man remains a hard problem for AI agents and humans alike. This is primarily because of the enclosed nature of the game space and the four-to-one ghost ratio. Simply trying to keep a certain distance from the ghosts is likely to end up with Ms. Pac-Man being trapped. Understanding how the ghosts will react to in particular situation, and so avoiding being trapped, is the key to survival in Ms. Pac-Man.

D. High Scores

The highest published score for an AI playing the original Ms. Pac-Man is 44,630 [5], using the screen-capture framework. This score was the maximum of 100 games, with level six being the highest reached. This would be considered a good score for a human. The best human players can reach scores in excess of 900,000, clearing more than 130 levels[3]. The highest score recorded by the Ensemble agent is currently 162,280 at level 24. This result was achieved while recording a video and is not part of the experimental data. The video is available on YouTube[4].

Ms. Pac-Man was recently released on Steam. The leader board would suggest 30,000 to be a reasonable average score for a human. In discussion with Patrick Scott Patterson—a video game advocate, journalist and record holder—Mr. Patterson suggested that six-figure scores were rare, and that only a handful of players in the world are capable of playing the game at this level. The current official world record is 933,580, set by Abdner Ashman in 2006. Only five people have officially reached over 900,000 points.

In their paper *Hybrid reward architecture for reinforcement learning* [6], van Seijen *et al.* describe an agent for the Atari 2600 version of Ms. Pac-Man. The agent they describe is very similar to the Ensemble agent described in this paper, but they use many more voices and the voices are learned rather than hand-coded. Their agent is effectively unbeatable, but the Atari 2600 version of Ms. Pac-Man is a lot simpler than the arcade version.

II. Monte-Carlo Tree Search in Ms. Pac-Man

MCTS has been applied to Ms. Pac-Man before. In their paper *Monte-Carlo Tree Search In Ms. Pac-Man* [7], Ikehata and Ito describe a MCTS agent for the screen-capture framework Ms. Pac-Man. For the MCTS simulations, Ikehata and Ito used a simplified model of the game. Despite using this simplified model, the agent entered and won the 2011 IEEE Ms. Pac-Man competition.

In 2014, Pepels et al published their paper *Real-Time Monte Carlo Tree Search* [8]. The paper describes an MCTS agent for the Ms. Pac-Man vs. Ghost Team framework. In this framework, agents have only 40ms to choose a move, so the team use depth-limited roll-outs and branch re-use to achieve competitive play.

[1] http://csee.essex.ac.uk/staff/sml/pacman/PacManContest.html
[2] http://www.pacmanvghosts.co.uk/
[3] https://www.twingalaxies.com/game/ms-pac-man/arcade
[4] https://youtu.be/Y9YazqWaEAM

III. Ms. Pac-Man vs. Ghost-Team

Initial experiments for this project were done using the Ms. Pac-Man vs. Ghost-Team framework. This led to some useful insights and a very capable agent—usually reaching the global time-limit at around level 12, often without losing a life. The agent played especially well against highly predictable ghosts, such as the aggressive ghost team, where it could group the ghosts together, eat a power-pill and then eat all the ghosts in quick succession.

To find out how the agent would fare against a top-ranking ghost team, we contacted the author of the *Memetix* ghost team, Daryl Tose, who very graciously sent us his code. As it turned out, despite the *Memetix* ghost team being almost completely deterministic, and the Ensemble agent being very capable against predictable ghosts, the agent rarely got past the first level. The conclusion we made was that however strong the Ms. Pac-Man agent is, the ghosts can always win if they work together as a team.

IV. JAMES

JAMES was written from the ground up to be an object-oriented Ms. Pac-Man emulator. Large sections of the code came from the ArcadeFlex project[5][6], an automated Java port of the Multi Arcade Machine emulator (MAME)[7]. The code was constructed as a core emulator, with a full emulator built around it. The core emulator emulates the CPU, RAM and I/O. The full emulator adds windowing, graphics and keyboard support. This allows the core emulator to be used as a forward model not tied to the 60 frames per second of the full emulator.

Agents interact with the game via the Game API. Game state information is obtained by interpreting the contents of specific memory locations within the emulator. Actions are performed by setting the values of the memory-mapped I/O ports. The Game class abstracts away from these low-level operations.

A graph data structure was created for maze-based queries. All-pairs tile distances were pre-calculated using the Floyd-Warshall algorithm [9]. In addition to the all-pairs distances, every tile stores the distance to every other tile for each available move. These directional distances were also pre-calculated, this time using A* search with the Floyd-Warshall computed distance as the heuristic.

Using the emulated Ms. Pac-Man code as a forward model is extremely accurate, but very inefficient. An alternative forward model, the *simulator*, was created.

V. The simulator

The simulator is a Java-native partial model of the game. A lot of work went into making the simulator as accurate as possible, especially with regard to ghost behaviour. Although the simulator it is not 100% accurate, it is generally accurate enough if synchronised with the emulator before each use.

[5]https://www.facebook.com/arcadeflex/
[6]https://github.com/georgemoralis/arcadeflex029
[7]http://mamedev.org/

Fig. 2: Opposing opinions

The simulator is very fast compared to the emulator, more than making up for the loss of accuracy.

A simple test was conducted to gauge the relative speeds of the emulator and simulator forward models. In this test, Ms. Pac-Man travels from the bottom-right corner of maze one to the bottom-left corner. This is a straight path, with pills, that takes 198 frames. In real-time, at 60 frames per second, that is just over three seconds. Averaged over 1000 runs, the emulator took 23.1ms; 137 times faster than real-time. The simulator took just 0.2ms; 16,417 times faster than real-time and 119 times faster than the emulator forward model.

VI. Ensemble Agent for Ms. Pac-Man

For the Ms. Pac-Man agent, the tasks were defined as:

- **Eat pills.** This is primarily a long range goal to clear the level.
- **Eat fruit.** This is a medium range goal to collect extra points.
- **Eat ghosts.** This is another medium range goal to collect extra points.
- **Avoid ghosts.** This is a short or medium range goal, depending on whether Ms. Pac-Man is escaping from a close-range ghost or avoiding being trapped.

It became apparent during initial experiments that simply having each voice offering its preferred move at any given point lead to a lot of deadlocks. The voices would often have opposing opinions due to the completely disparate nature of their goals. In Figure 2, Ms. Pac-Man is approaching a junction with three options: UP, LEFT or DOWN. The pill eating voice will vote to go DOWN, the ghost eating voice will vote to go LEFT and the fruit eating voice will vote to go UP. No reward is worth dying for, so the ghost avoiding voice will veto DOWN, leaving a deadlock between UP and LEFT. The move could be picked at random, or the arbiter could be crafted with some domain knowledge to make a more informed decision.

The final Ensemble Agent uses a technique similar to fuzzy logic, where each voice *rates* each of the available moves according to its own metric.

Using the same scenario as in Figure 2, with DOWN vetoed by the ghost avoiding voice, the other three voices need to present their ratings for UP and LEFT. As the voices are all distance based, the ratings will be the inverse of the distance to the goal in each available direction. If the voices are weighted equally, the resulting move would be UP. An approximation of the calculation can be seen in Table I.

	Pill eater	Fruit eater	Ghost eater	Sum (approx.)
UP	1/7	1/3	1/24	1/2
LEFT	1/7	1/24	1/4	2/5

TABLE I: Calculating move values

This solution is far less likely to lead to a tie-break situation, and it is also more flexible in terms of weighting each voice's opinion. In the above example the agent decided to go UP because the fruit is closer than the edible ghost, but it was close. Generally, there is likely to be more chance of eating the fruit in the future than the ghost, so LEFT would probably have been a better choice. Weighting the ghost eater higher than the fruit eater would have changed the decision to LEFT. Rating the pill eater low, because pills are relatively low value and static, would likely make the agent head towards the fruit after eating the ghost. The arbiter never actually targets anything, or makes any sort of plan. It simply chooses the highest combined-rated move at any given point.

The final Ensemble Agent for Ms. Pac-Man in *JAMES* is composed of four voices, with an arbiter taking the opinions of each voice and combining them to make the final decision.

- **Ghost Dodger.** Avoiding ghosts is the most important aspect of the game, and is also the hardest to do, computationally. It is the average result of sparse sampled, depth-limited roll-outs. This voice is discussed in detail in the next section.
- **Pill Muncher.** This voice rates each move as the inverse of the tile distance to the nearest pill in that direction. Pills near ghosts are artificially made to look further away, meaning that the pill muncher will rate safe pills higher than those with ghosts near by.
- **Fruit Muncher.** This voice has no opinion unless there is a fruit bonus on the screen. If a fruit is on the screen, the voice will attempt to intercept it. It rates the available moves as the inverse of the tile distance to the fruit.
- **Ghost Muncher.** This voice only has an opinion if Ms. Pac-Man is energised. This voice uses a similar sparse sampling technique as the Ghost Dodger voice, but it could also simply rate each move based on the distance to the nearest edible ghost. The sparse sampling technique allows for elegant behaviour such as intercepting ghosts, rather than simply chasing them.

A. Ghost Dodging

The final Ghost Dodger algorithm uses the simulator for depth limited search, and rates each move based on sparse random sampling. This algorithm is closely related to the averaged depth-limited search technique we applied to the game 2048, as demonstrated at the IEEE CIG2014 conference in Dortmund [10].

The algorithm is given 10ms in which to make random depth-limited, Monte-Carlo style samples through the maze. Every time it reaches its depth limit of 8 without dying, the initial move's score gets incremented. At the end of the 10ms, or it has found enough safe paths to be sure a move is safe, the voice returns its rating for each move, based on how many times it reached the depth limit. The depth of 8 was chosen as a trade-off between depth and the number of samples possible in the time-frame. For this algorithm, a *move* is a straight line from the current position to the next corner or junction. This simplifies the algorithm as Ms. Pac-Man's direction does not need to be re-calculated mid-move for cornering.

Because the simulator is not 100% pixel-perfect, it may determine a path to be safe when in the real game it is not. This usually occurs if there is a ghost very close. This becomes a real problem when deciding whether or not to clear a path of pills. Ms. Pac-Man pauses for a single frame when she eats a pill but the ghosts do not, so a following ghost will be faster. This quite often leads to situations where the simulator determines a path to be safe to traverse, only to realise its mistake when it is too late.

B. Arbitration

As previously discussed, the ensemble voices offer ratings for each available move. It is the job of the arbiter to combine the ratings into a decision. Voices associated with risk, such as the Ghost Dodger, are treated as multiplicative. Voices associated with reward, such as the Fruit Muncher, are treated as a sum. The overall value of a move is the sum of its rewards multiplied by the risk factor of that move.

In general, the first k voices of the total n voices are multiplicative, with the remainder being a summation. The value of each move m for each voice V_i is multiplied by weight W_i. The product of all the risk-based voices is calculated to give us a relative measure of risk. In this case, 0 indicates certain death and 1 indicates no risk.

$$RISK_m = \prod_{i=1}^{i=k} V_{i,m} W_i \tag{1}$$

We also need to calculate the reward associated with each move. This is done by summing all the move values for each of the reward voices:

$$REWARD_m = \sum_{i=k+1}^{i=n} V_{i,m} W_i \tag{2}$$

To calculate the final vector of move ratings R, we multiply these two factors. This ensures that any reward, no matter how large, will be nullified if the risk is too great.

$$R_m = RISK_m \times REWARD_m \quad (3)$$

The arbiter simply chooses the move corresponding to the highest valued R_m. If more than one move have the highest rating, the arbiter will select at random from the highest rated moves.

In this project the weights were hand-crafted and tuned until the behaviour was deemed good enough. These weights are unlikely to be optimal, so weight optimisation is a potential area for improvement.

1) Evolving Weights: Instead of hand-crafting the weights, as a proof of concept, a simple one-plus-one evolutionary strategy was used to optimise the weights of the voices. A baseline score was recorded over 100 games, then the weights were adjusted by a small random amount. Another 100 games were played with the new weights. If the new weight were kept if the average score improved. This was done 1000 times, for a total of 100,000 games. An early version of the Ensemble AI was used for this experiment, and it was able to increase the ability of that player. In this experiment, a noticeable increase in average score of approximately 30% was achieved. Unfortunately this technique takes a very long time, and it was not used in any of the final agents. In their paper *The N-Tuple Bandit Evolutionary Algorithm for Game Agent Optimisation* [11], Lucas et al describe an algorithm for optimising N-Tuple values for noisy and expensive problems. This approach could achieve better results quicker than the simple one-plus-one approach.

2) Dynamic Weights: Dynamic weighting is also a possibility, combining the Ensemble arbiter with a finite-state machine. Using such a technique would allow the agent to adjust the voice weights depending on the situation. This could be of use in, for example, stealth games. One set of weights could be used during stealthy missions, or portions of missions, while another set of weights used in situations where the player's stealth has been compromised. Dynamic weighting could also be used in general video game playing (GVGP) [12], especially if combined with general-purpose components.

VII. Monte-Carlo Tree Search

The MCTS agent was developed to set a high bar, and to demonstrate what a purely deliberative agent was capable of. Despite using the simulator forward model, and hence no knowledge of fruit, the MCTS agent manages to play to a very high standard.

A. Deliberation and Double Checking

In order for a our MCTS agent to perform well, it needs time to deliberate. In real-time games this is tricky. If the MCTS agent was trying to play synchronously with the game, a decision would need to be made every 16ms. Our simple implementation of MCTS was incapable of even playing the game, let alone play it well.

In order to solve the problem, we allow the MCTS agent to utilise the time it takes to get to its next target to run the simulations of what will happen when it gets there.

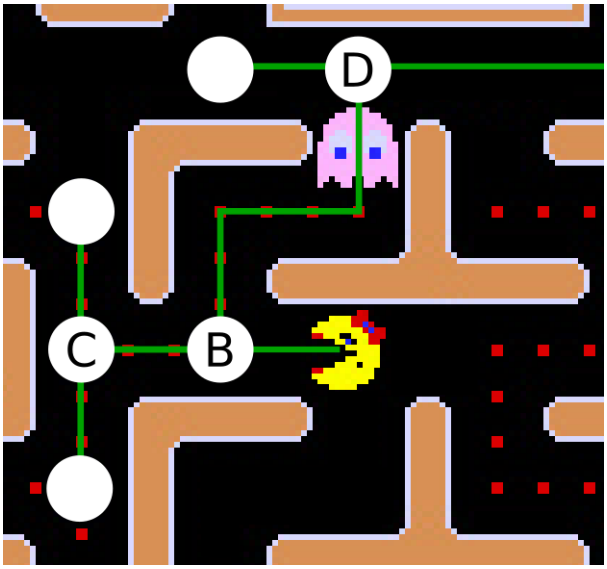


Fig. 3: MCTS paths

In figure 3, the agent is at point A and has committed to traveling to point B. Path AB has already been simulated and checked using the emulator forward model, so the agent can be almost certain that it is safe. During the time it takes to reach point B, the MCTS algorithm is running simulations from point B onwards.

The instant Ms. Pac-Man enters the tile associated with point B, the MCTS algorithm is stopped and the best move from point B is selected. In figure 3, BD always leads to death, so BC is selected. The agent will then double check BC using the emulator and commit to that move. The MCTS algorithm will start simulating moves from point C onward. If something unexpected happened and the emulator check shows a problem, the agent will cancel that move and go back; in this case, towards point A. The MCTS algorithm will be restarted from point A.

Using this travel time for deliberation allows the MCTS algorithm to visit each move hundreds, if not thousands of times.

B. Selection Phase

Preliminary experiments with exploration verses exploitation showed that with high exploration, and hence highly symmetric trees, the agent did not perform so well as with high exploitation. The symmetric and asymmetric agents both scored roughly the same on average, but the asymmetric agent's average level reached was 15, as opposed to the 10 of the symmetric agent[8].

From watching two versions of the MCTS agent playing, it could be seen that the symmetric agent starts off well, and does a great job of eating the edible ghosts, but when the level is almost complete—especially if there are two small, separate clusters of pills, the agent doesn't seem to know what to do.

[8]These results were obtained using an early version of the agent. The final MCTS agent plays much better, but the findings on symmetry still hold true.

There is very little reward available but lots of ghost death to be avoided. These situations lead to the agent getting stuck in a local optimal loop, going back and forth avoiding ghosts, but not getting the level complete. If the agent gets pushed by the ghosts close enough to one of the clusters of pills, it will take them. As soon as only one cluster of pills remains, the agent can see the win and will go for it.

In contrast to this, the asymmetric MCTS agent does not seem to do as good a job of eating the ghosts, and so scores fewer points, but it doesn't get stuck in local optima as often or for as long. The asymmetric nature of the trees means the agent is more likely to be able to see a winning path through two distant clusters of pills.

The average reward of a given move is used in the selection phase in place of the number of wins. As this reward is based on the score delta of the move, the exploitation value tends to be very high. To counter this, and create a better balance of exploration verses exploitation, am exploitation factor of 2000 is used.

C. Simulation Phase

Pure MCTS makes random, legal moves until an end-state is reached. While this approach works well for zero-sum games, for an arcade game like Ms. Pac-Man the only true end-state is GAME OVER. We could use intermediate end-states, such as when the agent is killed or the level is completed, but losing a life is also bad, and finishing a level is unlikely to happen making random moves.

Full roll-outs are difficult to assess, being as the vast majority of them will be death. Even if we use an evaluation function with a partial roll-out, the roll-out is likely to end in death. Another issue with evaluation functions is that they will heavily bias the roll-outs. If, for example, the evaluation function rewards distance from ghosts, the MCTS algorithm will avoid ghosts and end up being trapped just like a distance rule-based ghost avoidance algorithm.

The final, and best performing, MCTS agent has no roll-out phase at all. Once a new leaf is generated in the expansion phase, the current score delta is returned for back-propagation. The only cases where the score is not simply returned is when either the agent is killed or the level is complete. In the case of a death, the reward (score) is divided by 10. This has the effect of guiding the selection away from that part of the path, but without poisoning the entire branch, as was the case when the reward was set to zero on death. In the case of level completion, an extra 1000 points are added to the reward. That sounds low, but it is enough to get the agent going in the right direction without taking unnecessary risks to get the win.

D. No Emulator Checking

For the experiments where the agents cannot access the emulator, the MCTS agent did not do so well. Because it is always thinking one decision point ahead, it relies on the emulator to ensure the current path is safe. Removing this accurate checking mechanism proved to be a problem for the MCTS agent. The less than 100% pixel-perfect simulator can sometimes regard a path as safe when it is not; when a ghost is following Ms. Pac-Man very closely, for example. Just a single pixel discrepancy between the simulator and the actual game can result in completely different decisions from the ghosts, especially at close-quarters.

In order to have the MCTS agent able to perform reasonably well without the emulator, it needed to be able to react to its mistakes. This was achieved at the expense deliberation time by changing the algorithm to return a decision at each new tile, rather than at each corner or junction. The actual MCTS algorithm still used the corners and junctions as nodes when expanding the tree, but the tree gets rebuilt each time the agent enters a different tile. This allows the agent to effectively change its mind half way along a corridor, if a danger becomes apparent.

VIII. Experiments

The experiments themselves were very simple. Each agent plays 100 games at normal speed. The results of these experiments are then compared on both score and level reached. The level reached being used as a measure of the survivability of the agent. All experiments were run on a single desktop computer.

- **Machine**: 2011 Dell OptiPlex 790
- **CPU**: Intel Core i7-2600 running at stock 3.4GHz
- **RAM**: 16GB DDR3 running at 1333MHz
- **OS**: Linux Mint 17
- **JVM**: Oracle Java 7

Variations of the two main agents, MCTS and Ensemble, were used for comparison and fairness. Along with the 'best' versions of each agent, experiments were run with and without access to the emulator forward model. The MCTS agent makes far greater use of the emulator, so it is to be expected that this agent suffers more for having it removed. Hopefully the experiments show that the emulator is very useful, not a necessity.

Experiments were also run where the Ensemble does not include the fruit munching component. The fruit bonuses are not included in the simulator, so the MCTS agent has no knowledge of the fruit. This gives the Ensemble agent an unfair advantage in terms of point scoring. Turning off the fruit component removes this advantage.

IX. Results

Both agents use the emulator as a short-range, accurate forward model. The MCTS uses the emulator to double-check before it commits to a path, and the Ensemble uses the emulator to pre-filter the available moves to remove certain-death moves. The Ensemble also uses it as double-check for immediate death on the current move.

The MCTS agent reached level 13 in all 100 game played and reached level 21 in 68 of the 100 games. In terms of survivability, the MCTS agent is very strong. The main weakness of the MCTS agent is its relatively poor points

scoring. This is primarily due to the lack of knowledge of the fruit bonuses.

Because the MCTS agent does not have any knowledge of the fruit, a version of the Ensemble agent without the fruit muncher voice was also tested. In this configuration, the MCTS agent and the Ensemble agent score almost identical average scores.

The MCTS agent has greater survivability than the Ensemble, with an average level reached of 20.38 to the Ensemble's 18.67. That the Ensemble agent manages to score equally well in fewer levels than the MCTS agent, even without fruit, suggests that the Ensemble agent is slightly better at capturing ghosts than the MCTS agent.

Adding the fruit component to the Ensemble resulted in a 26% increase in maximum score, with a 33% increase in average score. This increase in score did come with a very small drop in survivability, with the average level reached dropping from 18.67 to 18.37, less than 2%.

A. Results With emulator

As can be seen in Table II, when compared to the MCTS agent the Ensemble agent scores significantly better. Most of the extra points seem to come from the MCTS agent's inability to effectively capture fruit. When the Ensemble agent has the fruit muncher voice disabled, there is no significant difference in scoring compared to the MCTS agent.

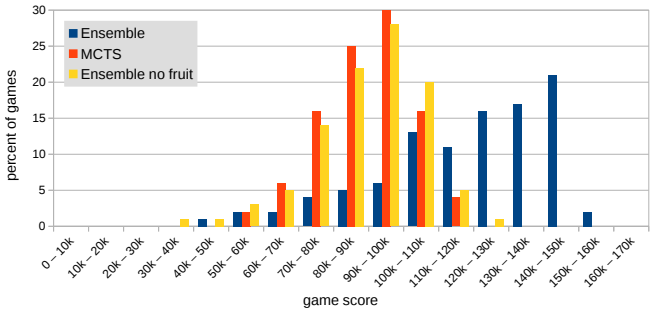


Fig. 4: Chart of score ranges reached with emulator

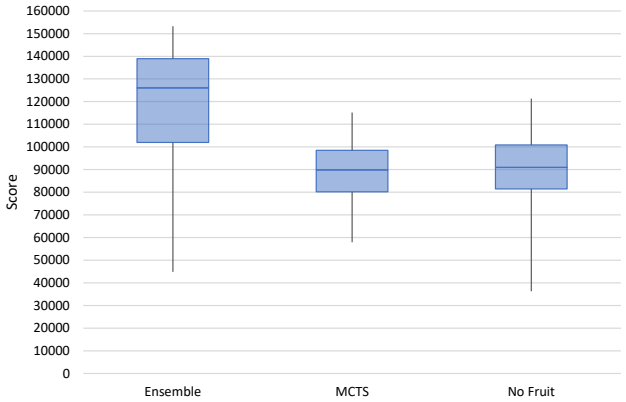


Fig. 5: Box chart of agent scores with emulator

Agent	Minimum	Maximum	Mean	Std. Err.	p-value
Ensemble	44860	153280	118610	2477	$< .0001$
MCTS	57920	115180	89278	1286	
Ensemble no fruit	36310	121360	89095	1547	.9276

TABLE II: Comparison of scores with emulator

Table III shows the levels reached by each agent. These figures give a measure of the agents' survivability. Compared to the Ensemble agent, the MCTS agent is significantly better at surviving. The fruit munching voice does not significantly affect the survivability of the Ensemble agents.

Agent	Minimum	Maximum	Mean	Mode	p-value
Ensemble	6	25	18.37	21	
MCTS	13	24	20.38	22	$< .0001$
Ensemble no fruit	8	24	18.67	21	.5852

TABLE III: Comparison of levels reached with emulator

B. Results Without emulator

In terms of both scoring and survivability, all agents were significantly worse off for having the emulator forward model removed (p-value $< .0001$), but he MCTS agent was the most dramatically affected. The MCTS is more heavily dependent on the emulator for accurate checking of paths. The ensemble uses the emulator much less; only for pre-filtering moves, and as an extremely short range (eight frames) safety check.

The full Ensemble's performance drops by about 23% for both average score and average level reached. The Ensemble without fruit loses about 19% in average score, and 17% in average level reached. The MCTS agent, however, loses a huge 48% in average score, and 52% in average level reached.

Table IV shows the scores for each agent over 100 games. Compared to the MCTS agent, both of the Ensemble variants are much higher scoring.

Agent	Minimum	Maximum	Mean	Std. Err.	p-value
Ensemble	16760	145510	91167	2848	$< .0001$
MCTS	3930	92660	46672	2028	
Ensemble no fruit	26200	115350	72586	2064	$< .0001$

TABLE IV: Comparison of scores with emulator

In Table V we can see that the MCTS agent is now significantly worse at surviving than the Ensemble agent, and that removing the fruit munching voice does not significantly change the survivability of the Ensemble.

Agent	Minimum	Maximum	Mean	Mode	p-value
Ensemble	3	22	14.19	16	
MCTS	1	20	9.83	7	$< .0001$
Ensemble no fruit	6	23	15.49	21	.0642

TABLE V: Comparison of levels reached without emulator

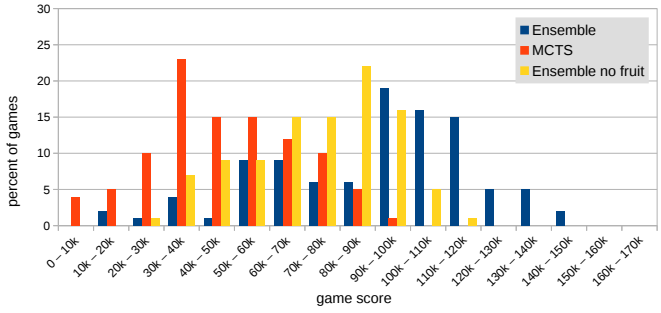


Fig. 6: Chart of score ranges without emulator

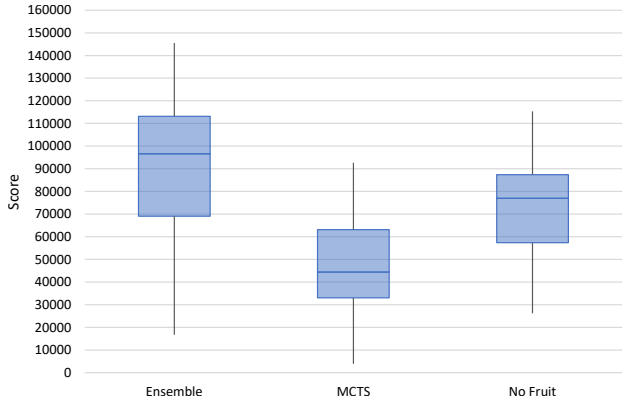


Fig. 7: Box chart of agent scores without emulator

X. Conclusion

This paper described the use of an Ensemble Decision System to create a Ms. Pac-Man agent. It also described a new framework that allows Java controllers to be written for the original Ms. Pac-Man arcade game. Using this framework, and a powerful simulator forward model, we were able to create very high scoring agents.

The Ensemble agent scored a world record AI score for Ms. Pac-Man, outperforming a MCTS-based monolithic agent based on the same system and forward model.

The Ensemble decision systems show great potential as efficient and flexible alternatives to monolithic agents, and also as lightweight augmentations to existing systems. Adding fruit awareness to the MCTS agent, for example. This could be done without interfering with the MCTS algorithm in any way, only potentially altering the chosen move.

XI. Future Work

The simple arbiter could be replaced by something more sophisticated and dynamic; possibly a trained neural network or a genetic algorithm to learn a strategic sense of the game. The experiments evolving the voice weights of the ensemble were mildly successful, but slow and tedious. It does leave open the possibility of using better optimisation algorithms.

The results for Ms. Pac-Man are very good, but it is only one game. We would like to see Ensemble Decision Systems applied to GVGP problems, to get an understanding of how flexible they can be, and if multi-purpose, or reusable, voices can be created.

XII. Code

The code used in this project can be downloaded from GitHub at https://github.com/philrod1/james

Acknowledgements

The authors would like to thank:
George Moralis and the ArcadeFlex developers.
Scott Lawrence for the Ms. Pac-Man disassembly.
Jamey Pittman for the Pac-Man Dossier

References

[1] B. Dasarathy and B. V. Sheela, "A composite classifier system design: Concepts and methodology," *Proceedings of the IEEE*, vol. 67, no. 5, pp. 708–713, May 1979.

[2] D. Ferrucci, E. Brown, J. Chu-Carroll, J. Fan, D. Gondek, A. A. Kalyanpur, A. Lally, J. W. Murdock, E. Nyberg, J. Prager *et al.*, "Building watson: An overview of the deepqa project," *AI magazine*, vol. 31, no. 3, pp. 59–79, 2010.

[3] Y. Chen, J. E. Argentinis, and G. Weber, "Ibm watson: How cognitive computing can be applied to big data challenges in life sciences research," *Clinical Therapeutics*, vol. 38, no. 4, pp. 688 – 701, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0149291815013168

[4] P. Rohlfshagen, J. Liu, D. Perez-Liebana, and S. M. Lucas, "Pac-man conquers academia: Two decades of research using a classic arcade game," *IEEE Transactions on Games*, vol. PP, no. 99, pp. 1–1, 2017.

[5] G. Foderaro, A. Swingler, and S. Ferrari, "A model-based cell decomposition approach to on-line pursuit-evasion path planning and the video game ms. pac-man," in *2012 IEEE Conference on Computational Intelligence and Games (CIG)*, Sept 2012, pp. 281–287.

[6] H. van Seijen, M. Fatemi, R. Laroche, J. Romoff, T. Barnes, and J. Tsang, "Hybrid reward architecture for reinforcement learning," in *NIPS*, 2017.

[7] N. Ikehata and T. Ito, "Monte-carlo tree search in ms. pac-man," in *2011 IEEE Conference on Computational Intelligence and Games (CIG'11)*, Aug 2011, pp. 39–46.

[8] T. Pepels, M. H. M. Winands, and M. Lanctot, "Real-time monte carlo tree search in ms pac-man," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 6, no. 3, pp. 245–257, Sept 2014.

[9] S. Warshall, "A theorem on boolean matrices," *J. ACM*, vol. 9, no. 1, pp. 11–12, Jan. 1962. [Online]. Available: http://doi.acm.org/10.1145/321105.321107

[10] P. Rodgers and J. Levine, "An investigation into 2048 AI strategies," in *2014 IEEE Conference on Computational Intelligence and Games, CIG 2014, Dortmund, Germany, August 26-29, 2014*. IEEE, 2014, pp. 1–2. [Online]. Available: http://dx.doi.org/10.1109/CIG.2014.6932920

[11] S. M. Lucas, J. Liu, and D. Perez-Liebana, "The N-Tuple Bandit Evolutionary Algorithm for Game Agent Optimisation," *ArXiv e-prints*, Feb. 2018.

[12] J. Levine, C. B. Congdon, M. Ebner, G. Kendall, S. M. Lucas, R. Miikkulainen, T. Schaul, and T. Thompson, "General video game playing," *Dagstuhl Follow-Ups*, vol. 6, 2013.

Integrated Balancing of an RTS Game: Case Study and Toolbox Refinement

Mike Preuss, Thomas Pfeiffer
ERCIS, WWU Münster
Department of Information Systems
mike.preuss@uni-muenster.de,
thomas-pfeiffer@live.de

Vanessa Volz
TU Dortmund University
Department of Computer Science
vanessa.volz@cs.uni-dortmund.de

Nicolas Pflanzl
ERCIS, WWU Münster
Department of Information Systems
nicolas.pflanzl@uni-muenster.de

***Abstract*—A recent publication suggested an integrated process for game balancing by means of a combination of manual and automated balancing approaches. In this work, we refine the approach and show how to apply it to a "real game", not a simple prototype, namely to an open source clone of the real-time strategy game "Red Alert". Specifically, we are interested in how to utilize the outcomes of automated balancing to inform and direct our iterated approach. This enables us to follow and explore several diverse balancing solutions at the same time, which should in turn improve performance and reliability of the balancing process. We are not primarily interested in the determined solutions, but in the process that can easily be adapted to other games and balancing goals.**

***Index Terms*—balancing, RTS games, process**

I. Introduction

Game balancing is defined as the "process of systematically modifying parameters of game components and operational rules in order to determine satisfactory configurations regarding predefined goals" [3, p. 1]. It is well known to be a non-trivial issue that is especially necessary but also challenging in later phases of game design to ensure, e.g., the fairness of the final game [14]. While the prevalent approach in the industry still mostly consists of manual trial-and-error methods based on the tacit knowledge and experience of the game designer, there have been a number of approaches that show the potential of (at least partially) automated balancing, employing AIs that mimic the behavior of human players, and utilizing algorithms for optimization.

Recently, Beyer et al. have proposed an *integrated* process for game balancing that combines the advantages of both manual and automated balancing activities [3]. Based on its application to a simple test game, the authors demonstrate that automation can be beneficial, but also has certain downsides. We continue this line of research by reporting on the results of a case study conducted using a more complex real-time strategy (RTS) game to investigate how the results of automated balancing can be used to inform and direct the entire balancing process. This represents a conceptual shift from uncoordinated alterations between manual and automated activities towards a more structured approach where automated balancing serves the *exploration* of meaningful parameter combinations, whereas manual game-testing provides *exploitation* by applying smaller changes to obtain feasible solutions.

The main contribution of this paper lies in an examination of how automated balancing via optimization can be conducted in practice, i.e. how the problem complexity can be suitably reduced. Furthermore, we investigate how it is possible to integrate automatic balancing into the overall balancing process that is still very much based on intuitive design decisions and heuristics. To that extent, we modify the process proposed in [3] to enable a more dynamic integration of manual and automatic balancing. More concretely, we especially add assessment simulations and correlation checks to our toolbox in order to inform the manual steps. Furthermore, we also divide the preparatory steps into subtasks and show how these can be executed for a 2-player RTS game. Ultimately, we hope to be able to transfer the experiences from this case study to a more general level.

The remainder of this paper is structured as follows: First, Section II gives a brief overview of the fundamentals of game balancing and presents the state-of-the-art approaches employing some degree of automation. Afterwards, Section III describes OpenRA, the test game that was employed in the case study. The applied balancing process and the modifications that were made to the proposal by Beyer et al. are outlined in Section IV, and the results of several experiments conducted as part of the case study are reported in Sections V and VI. Finally, the paper closes with the conclusions.

II. Related Work

Game balancing is a process that tunes a game to fulfill one or multiple design goals, such as fairness, average game duration, or difficulty (cf. [14]). In the context of RTS games, fairness, in the sense that the winner is determined by player skill, is one of the most prominent points of contention. It is often targeted via patches in popular RTS games[1], discussed by industry professionals[2] and has also been the subject of research. Mahlmann et. al target it in consecutive work where they propose an RTS-specific representation language and several evaluation functions to find balanced game configurations. Specifically, they try to find games where (1) the units are complementary, (2) the outcome is uncertain for a long time,

[1] http://liquipedia.net/starcraft2/Patches

[2] GDC talk: Real-Time Strategy Game Balance (https://www.gdcvault.com/play/1022530/Real-Time-Strategy-Game)

978-1-5386-4359-4/18/$31.00 ©2018 IEEE

or (3) the lead changes multiple times. To this end, they propose three different fitness functions that are computed based on data collected from the gameplay of AI agents. As in our case, they optimise these fitness functions using an evolutionary algorithm. While they describe interesting results, they also find that the proposed evaluation methods vary with the specific AI agent employed in the simulations. They recognise that in order to obtain meaningful measurements, the AI would need to behave human-like in some aspects, which would need to be determined as well. As a potential solution to this issue, we propose an integrated approach with human designers and play-testers, which counteracts overfitting to AI characteristics.

In general, using AI gameplay simulations to evaluate some aspects of a game which are then optimised using an evolutionary algorithm is a relatively wide-spread approach in game balancing. It has been applied to card games [18], Ms PacMan [11], tower defence games [1] and many more. However, the examples we know all conduct full-automatic balancing and thus also lack human behaviour modeling. Moreover, they are generally applied to games with only a small number of permutable parameters, and a relatively small search space is of course beneficial for any kind of optimization algorithm. Additionally, many of these examples are short, relatively simple games, so that the implementation of an AI is not prohibitively difficult and the simulation is not excessively computationally expensive.

The research fields around procedural content generation [17] and dynamic difficulty adaptation are related to AI-assisted game balancing, as all need some form of automatic evaluation of the degree to which specified design goals are satisfied. For example, in [7], the authors find that human players experience RTS games against AI opponents with dynamically adapting difficulty as most enjoyable. Such AI agents are proposed in [5, 12].

Work on gameplay evaluation, procedural content generation and dynamic difficulty adapation of course also exists in context of other genres, e.g. first-person shooters [4, 8] or role-playing games [16], and there are even some generalisable approaches [10]. However, the focus in this paper is on the balancing process itself instead of game evaluation, and specifically its integration with human designers and players. We will thus refrain from providing a detailed overview of these research areas.

Finally, there is a comparatively large corpus of work on mixed-initiative design, which also integrates automatic evaluations and human opinions. This includes map/level design for, e. g., RTS games (Sentient Sketchbook) [9], platformers [15] and mobile games [13]. However, these approaches mostly focus on said map/level creation and/or optimization, whereas we want to approach game balancing with a mixed-initiative concept.

III. Test Game: Open Red Alert

Rather than implementing our own game prototype, an existing game was chosen to examine whether our balancing approach is well-suited for published games with standard-to-high complexity (as opposed to toy games), and to show that (semi-)automated balancing can be plugged into an existing game. The choice of game was restricted to the RTS genre for three main reasons: 1) It has an especially high demand for balancing, as allowing the development of sophisticated and diverse strategies by players requires a deep and fair game as foundation. 2) The inherent complexity of RTS games with different unit types and high challenges for AI players makes them hard to balance. 3) The concept of balance is rather tangible for RTS games; establishing a fair and engaging setup of different factions is the main balancing challenge for most RTS games.

Fig. 1. Screenshot of the test RTS game Open Red Alert.

We determined the following requirements for potential game candidates: (1) *open source code access* to allow for the implementation of any necessary modifications (e.g., headless simulation runs), (2) *ongoing maintenance* so that help can be easily obtained should technical problems occur, and (3) *support for small changes* by the game architecture to enable a simple adaption of game parameters relevant for balancing without requiring the game to be recompiled. The three most suitable candidates that were discovered are MegaGlest[3], 0 A.D.[4], and OpenRA[5]. The choice was made in favor of OpenRA due to its parameterization via YAML files, which makes it easier to handle than MegaGlest, and the early development state of 0 A.D.

Compared to the original Command & Conquer: Red Alert (Westwood Studios, 1996) on which OpenRA is based, the following notable changes were made: the introduction of a Fog of War, a non-cheating AI (i.e., the AI does not get more information or resources than human players), and a re-balancing of factions. The non-cheating AI is advantageous for automated balancing as it likely approximates a human

[3]https://megaglest.org/
[4]https://play0ad.com/
[5]http://www.openra.net/

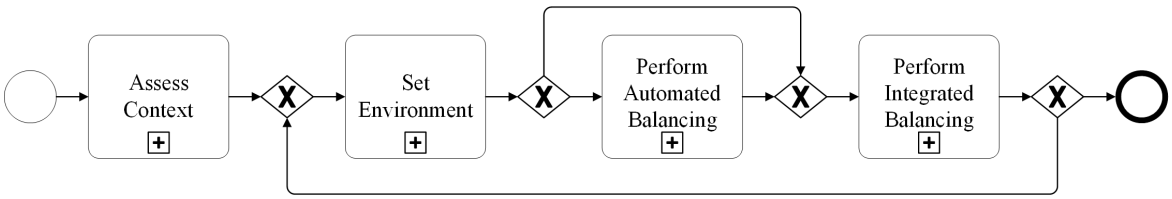


Fig. 2. The slightly modified new balancing process (cf. [3]), modeled in BPMN (*Business Process Modeling and Notation*). Plus signs indicate the existence of subtasks within the activity, X stands for an either/or decision. The main change is that automated balancing is inserted as a preparatory step for the integrated balancing phase that also partly relies on automation.

play style more closely than an AI that can cheat.

As the original, OpenRA is set in a fictional reality in which the death of Adolf Hitler in 1924 results in an alternate World War II fought between the Soviet Union and the allied nations of Europe. It is a typical RTS game in which players are tasked with destroying all of their opponent's buildings, to which end they must farm ore to obtain credits as a resource for creating units and buildings, and produce energy through power plants to allow existing buildings to operate. A screenshot of the game UI is shown in Fig. 1. The sidebar on the right provides controls for the creation of units and buildings. It furthermore shows information about resources; the player is currently in possession of 2900 credits and has a power surplus of 20. The resource ore is shown on the lower right, as well as an ore truck gathering it.

Of the requirements most important for balancing RTS games, the most prominent one from the perspective of players is arguably fairness. Other requirements such as richness (multiple meaningful strategies and no dominant strategy) and usefulness (meaningfulness of every unit/building in some context) exist as well, but are typically addressed through the concept of *intransitive superiority* [2, 19], also known as rock-paper-scissors, on the unit level.

For symmetric games with identical player setups (same units, buildings, symmetrical map), fairness is trivially obtained and—with the exception of non-deterministic mechanisms—the winner should be determined exclusively by player skill. However, RTS games are not like chess in this regard: their fascination comes to a large extent from replaying historical or fictional settings as ancient, fantasy, World War II, or science fiction settings. Asymmetry is therefore built into these games, be it in the form of asymmetric maps or goals, or, as most frequently encountered, factions with different units and buildings. As this is also the case for OpenRA, in the following we will exclusively focus on fairness under asymmetric conditions for competitive scenarios with two human players. Obtaining fairness is then the task of the balancing process that shall ideally respect any ideas about the game play as envisioned by the game designer as well as playability.

IV. Integrated Game Balancing Process

In [3], Beyer et al. propose a standard process model for game balancing that is intended to be valid across all games and genres. This model is *integrated* in the sense that it considers both *manual* and *automated* balancing activities as well as their iterative repetition towards some balancing goal. Nevertheless, its definition also allows for purely automated or exclusively manual instantiations of the balancing process.

In the proposal by Beyer et al., the role of automated balancing was rather unspecified and the interaction between manual and automated balancing unclear. Consequently, we suggest to modify this scheme by adding some automation also to the manual balancing phase, albeit with a more precise purpose. The modified process model is depicted in Fig. 2, using the BPMN standard (*Business Process Modeling and Notation*)[6] widely used in the Information Systems discipline for representation. In the following, we give a high-level description of the individual steps (activities) of this process set against the backdrop of RTS games in general before applying it to OpenRA.

Step 1: Assess context. The first activity consists of two subtasks. 1) *Define scope*: First, the game or game component to be balanced must be chosen. 2) *Define goals*: Second, the balancing goal must be specified in human-readable as well as machine-readable, computable form. The latter represents an objective function encapsulating the intentions of the game designer and is used to conduct automated balancing and to evaluate balancing success later on. The importance of this subtask lies in the fact that the concrete choice of a goal function strongly influences the characteristics of the game after balancing. After this step, all further activities are part of a cycle and may thus be repeated an arbitrary number of times.

Step 2: Set environment. The second activity consists of four subtasks. 1) *Choose scenario*: Having defined the scope, we next need to select levels, game mechanics, units, buildings, and other components that constitute the scenario to be balanced. This will typically entail the omission of some constituents of the game play, such as special units that are only used in single-player missions. 2) *Choose parameters*: Next, it must be decided which game parameters are considered out of bounds of the balancing endeavor (e.g., due to their presumed minor importance) and the values of these parameters must be set. This may become necessary if the amount of parameters otherwise becomes too large for an optimization-based approach and means that some aspects of

[6]www.bpmn.org Business process models are particularly employed for describing how computer systems and humans interact, in contrast to data flow models or algorithmic descriptions in computer science.

game objects are disregarded. To give an example, units may possess a speed parameter for all possible terrain types, but the disadvantage of complicating the balancing process with about 10 parameters more per unit may outweigh the advantage of diversifying unit behavior by far. 3) *Define parameter bounds*: For the chosen parameters, lower and upper bounds as well as possible step sizes must be defined. This is necessary for automated balancing, but can itself be considered an act of manual balancing. Consequently, the bounds should not be so tight that a large number of potentially interesting solutions is excluded, but also tight enough so that solution candidates are feasible with a high probability. 4) *Implement automated environment*: Finally, an environment that is capable of running automated games must be implemented. Ideally, the game can be run *headless* (i.e., without GUI) and with a quicker pacing than the normal speed, as this enables more simulations in shorter time. Furthermore, an AI that can act as a stand-in for human players is needed. This AI should be *complete* in the sense that it has access to and can exploit all game mechanics. For RTS games, this is usually not a problem as the existing AI has to deal with the same problems as the human player (controlling a whole faction) Nevertheless, there may still be technical issues because games are not usually intended to be played in a fully automated fashion without any human intervention. Thus, while it may be argued that the AI may never fully replace a (particular) human player, we presume that the advantages of automated test runs outweigh the disadvantage that the playing styles of AIs are more restricted than the ones of humans.

Step 3: Perform automated balancing. In the third step, an optimization algorithm needs to be chosen for automated balancing and a stopping condition must be defined. Here, the objective function specified for the balancing goal in the first step plays an important role again. We generally recommend evolutionary algorithms or related meta-heuristics as the problem itself is black-box, i.e., it depends on the results of a simulation. However, the ideal, perfectly balanced solutions is likely unknown and in addition difficult to find due to the uncertainty inherent to non-deterministic AI decisions or game mechanics (e.g., randomized damage values). As a result, the stopping criteria for the optimization algorithm will either be available time budget and/or detected stagnation. The outcome of this activity is a set of the best balancing parameter configurations found that are sufficiently different from one another with regard to some quantitative indicator.

Step 4: Perform integrated balancing. The final activity consists of two subtasks that can be executed in any order or number of repetitions as deemed necessary. The purpose of this is to validate solutions yielded by the automated balancing process and to adjust them with respect to the original intentions of the game designer if the playing experience deviates from the former. The two subtasks are as follows.

1) *Execute simulations*: Having refined some candidate solution through small adjustments of its parameter configuration the modified solution should be playtested automatically to assess its effect. For instance, performing 100 simulations before applying a change and 100 afterwards would facilitate examining its impacts on game play. We call this simple technique *assessment simulations*. 2) *Conduct manual playtest*: Possible motivations for conducting additional, manual playtests include validating a game configuration regarding design ideas, complementing AI-based evaluations if a divergence from human behavior is expected, or checking if a solution also leads to a satisfactory play experience for other scenarios than those chosen in the second step. Furthermore, it may also become obvious that a solution is already quite good but a small set of parameters appears problematic. These parameters can then be changed manually and tested again, essentially signifying a very limited and thus fast manual balancing process. Additionally, the playtests may be assisted by another round of simulations. As simulations only test one specific parametrization, they are much faster to execute than automated balancing.

V. Case Study: Preparation

Step 1: Assess context. According to the balancing process outlined in Sect. IV, the first activity is to assess the context, consisting of scope and goal definition. In case of OpenRA, we have a set of 39 different unit types, each possessing about 40 parameters. Doing an automated balancing run with more than 1000 real-valued variables seems impractical if not infeasible if a single simulation takes some seconds. Such an optimization run would probably require at least around 1 million evaluations, even in a noiseless environment. In the end, it is highly likely that even for the best solutions, some parameters would not be well adjusted and need to be corrected manually. There are certainly several ways how to deal with this problem; we suggest a bottom-up approach that first balances the most important units/parameters and is then extended step by step.

Define scope: from the available game modes (missions, skirmish), we select the 2-player skirmish games as this is probably the game setting that is played most often and thus the most interesting.

Define goals: our high-level goals are *fairness, richness*, and *tension*. More concretely, for OpenRA balancing we demand that: 1) a 2-player skirmish with different factions on a symmetric map should be fair (success largely depends on player skill), 2) all unit types must be relevant for the gameplay, and 3) matches should be close, meaning that both factions should loose about the same amount of troops, taking importance differences into account (a tank is more relevant than a rifleman). The latter is a very rough heuristic only and could be replaced or complemented by other conditions as the length of games.

Step 2: Set environment. In order to establish the balancing environment, we first need to *choose a scenario*. This entails selecting units, buildings, and maps to be considered. According to our bottom-up approach, we concentrate on the core game first, that is the set of units that are available early in the game. Besides the MCV (*mobile construction vehicle*) that is used to establish the base and then disappears from

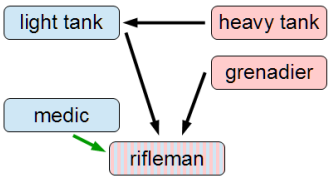


Fig. 3. Relations between combat units of the first iteration: backgrounds symbolize factions (blue=allied, red=soviet), units on the same line are functional counterparts, black arrows stand for a *is well suited against*, green arrows for *support*.

the game, we also need to add *ore trucks* in order to enable harvesting the main resource, ore. However, since these do not take part in battles (other than being destroyed), we thus just adopt their parameters from the default parameter set that is shipped with the game. Nevertheless, their speed and capacity of course has some influence on the game.

The most important battle units are displayed in fig. 3 along with their relations, namely the rifleman, the basic infantry unit that is available to both factions, the medic, the grenadier, the light tank and the heavy tank. This selection is a compromise between enabling a sophisticated gameplay and reducing the number of units to a minimum, according to expert judgement.

As these units have to be built and supported, we need to add 5 basic building types: construction yard, power plant, barracks, ore refinery and war factory. Additionally, a symmetric simple map without many obstacles is selected, as we want to avoid adding any bias in this step. Later on, resulting parameterizations will be validated on other maps as well.

Given the available units, we need to *choose the parameters* that will be subject to change, as well as *define parameter bounds*. As ≈ 200 parameters are still infeasible, we reduce their number by disregarding all parameters (e.g., separate movement speeds for different terrains) that, according to our assessment, do not have a large impact. Tab. I shows the 24 (the medic does not do damage) selected parameters and their relatively wide value intervals. This choice avoids restricting the available choices too much but keeps basic design ideas (e.g., tanks should have higher cost than infantry) intact.

TABLE I
UNITS AND PARAMETER BOUNDS FOR THE FIRST BALANCING ITERATION

Unit	Parameter	Shorthand	Min. Value	Max. Value
Rifle Man	Cost	C_R	20	200
	Reload Delay	R_R	5	40
	Damage	D_R	2	20
	Health	H_R	10	60
	Movement Speed	M_R	10	80
Medic	Cost	C_M	20	200
	Reload Delay	R_M	40	200
	Health	H_M	20	80
	Movement Speed	M_M	10	80
Grenadier	Cost	C_G	20	200
	Reload Delay	R_G	15	60
	Damage	D_G	10	50
	Health	H_G	20	80
	Movement Speed	M_G	10	80
Light Tank	Cost	C_L	200	1000
	Reload Delay	R_L	2	15
	Damage	D_L	2	15
	Health	H_L	100	450
	Movement Speed	M_L	10	120
Heavy Tank	Cost	C_H	500	2000
	Reload Delay	R_H	30	100
	Damage	D_H	25	80
	Health	H_H	500	1000
	Movement Speed	M_H	10	60

Next, we have to *prepare the automated environment* for running balancing optimization runs. Game parameter settings are read via console option or parameter file and the results are written to a log file and an SQL database. Automation features three main requirements:

- the game must run without GUI in the background
- the game speed should be as fast as possible
- an AI must be available that plays in a human-like fashion

In contrast to a game under development where these requirements can be implemented on the fly, the game code (C#) had to be changed in our case. By means of introducing a parameter that determines the length of a game frame, we managed to speed up headless game runs to about a factor of 40. Another important aspect of the automated environment is the availability of an AI that can play as stand-in for a human player. The AI engine of OpenRA is named *Hacky AI*, and it comes in different *instantiations*: Rush AI, Turtle AI, and Naval AI. In the following, we employ the Rush AI only, as it is the most versatile version. The AI is implemented via simple rule sets, but its behavior looks reasonably similar to a human playing style. By means of several comparison tests, we made sure that the AI behavior stays the same for headless, accelerated runs. If the GUI is switched off, we can speed up the game around a factor of 2.

VI. Case Study: Balancing Iterations

We are now going to run through the cycle of automated and integrated balancing (according to fig. 2) 2 times, with only slight changes to the environment in the successive iterations. Of course, 2 is not a fixed number. More iterations can be performed as needed. After every finished iteration, it is up to the user to decide if the current solution is satisfactory and the process can be stopped. However, 2 iterations already show what kind of difficulties are encountered and how we can add more game objects after being satisfied with the balancing of the restricted system. In the descriptions, prevailing decisions will not be mentioned again.

A. Setting Up Target Function and Algorithm

While the game is prepared now for automated parameter optimization, we also need to choose a target function and an optimization algorithm. Formally, this belongs to the *perform automated balancing* step of the first iteration. However, both choices will be kept fixed throughout the paper, and for reasons of clarity, we single it out here.

From the 3 goals named in sect. V, we choose the third (comparable losses of buildings and units mean close games) for setting up a target function as the first is partly included in it and the second (all unit types relevant) cannot be assessed very well automatically at this stage (we have a very restricted

TABLE II
PEARSON CORRELATIONS BETWEEN USER FEEDBACK AND CLOSENESS

Measurements			ρ
F_A	:	Fitness	-0.82
F_B	:	Fitness	-0.77
C_A	:	Fitness	-0.70
C_B	:	Fitness	-0.69
C_A	:	F_A	-0.82
C_B	:	F_B	-0.77

setup with only 5 unit types here, the second goal would become more important later on when more units are added). As buildings are much harder to destroy than units, we measure the absolute difference between players in both and weigh buildings with a factor of 5 (see equation (1), where D_{px} is the number of buildings destroyed by player x and K_{px} the number of units killed by player x).

$$f(D_{p1}, D_{p2}, K_{p1}, K_{p2]}) = 5 \times |D_{p1}, D_{p2}| + |K_{p1}, K_{p2]}| \quad (1)$$

This target function has been validated by a user playtest and survey, with 2 experienced (and similarly skilled) RTS gamers playing 15 games of the unmodified restricted version and rating each game on a scale from 1 to 10 for how much fun they had (F_A/F_B) and how close the game was perceived (C_A/C_B). These two are probably connected for RTS games. We assume that at least for experienced players, fun depends on the impression of fairness as a precondition. Table II shows the Pearson correlation of the fitness values computed on the resulting games (equation (1)) as well as the respective fun and closeness values according to the players. As the correlation values (fitness is minimized, therefore the negative correlation) are around 0.7 and higher, the test clearly indicates that the target function is reasonable, and we will thus stick to it in this work. This does not mean that there are no (possibly even better) alternatives.

Concerning algorithm choice, the environment (noisy objective function provided by a relatively slow simulation engine) strongly suggests a meta-heuristic approach. As a rule of thumb, doing 1000 simulations needs around 10 hours of computation time, putting us into the realm of (noisy) expensive optimization. We thus apply an Evolution Strategy (a simple Evolutionary Algorithm with deterministic selection, see [6]) with a rather small population and plus-selection (the current best individuals are only replaced if better ones are found). In order to cope with the noise, every solution candidate is evaluated 3 times and the median is then used as objective value.

Whereas algorithm runtimes do not allow for an extensive parameter tuning, we have tested several parameter sets and eventually agreed on the following 3 settings:

- Recombination is meant to generate original mixes from existing parent solutions, but also puts us in danger of losing diversity quickly. In order to keep different types of solutions in the pool, we generate only 20% of the offspring by applying uniform crossover, 80% are generated as clones of a single parent. All offspring individuals undergo mutation in any case.
- As opposed to many numerical optimization settings, small changes to the parameter values have often little effect in our context. We thus employ a uniformly distributed mutation of $\pm 5\%$ instead of a normally distributed one.
- A small population size enables rapid progress, a larger can keep more diversity (and thus more completely different solutions). After some tests with different sizes, we fix parent and offspring size to 6, resulting in a (6+6)-EA. Smaller population sizes quickly lead to quasi-fixation of all parameters after a few generations.

The resulting EA is implemented based on the Python library `evoalgos`[7].

TABLE III
MOST STRIKING PARAMETER DIFFERENCES AND GROUPING OF 10 BEST SOLUTIONS OF AUTOMATED BALANCING IN ITERATION 1

Solutions	$S_1, S_2, S_4, S_5, S_8, S_9$		S_6, S_7, S_{10}		S_{G1}	S_{G_2}
Parameter	Min	Max	Min	Max		
R_M	110	122	63	69	116	66
M_M	43	52	25	27	46	26
C_G	165	177	57	59	172	58
D_G	20	25	46	48	22	46
D_H	37	43	65	72	38	68
M_H	34	39	20	20	36	20

B. Iteration 1

Step 3: Perform automated balancing. After performing 5 optimization runs, a list of the best 10 overall solutions (S_1 to S_{10}) is compiled and manually (visually) clustered according to their differences in parameter values, as depicted in tab. III. With the exception of one solution that seems to be a mixture of both groups (discarded here), two distinct clusters are obtained; these are named S_{G1} and S_{G2}. The left part of the table shows the parameter ranges of the grouped solutions for the most different 6 parameters, the right part the rounded averages which form the starting point of the integrated balancing.

In what do the two groups differ? Interestingly, all 6 parameters belong only to 3 of the 5 units, namely medic (reload delay, movement speed), grenadier (cost, damage), and heavy tank (damage and movement speed). As the medic is an allied unit and the other two are soviet units, we presume that somehow the changes on the one side (if comparing S_{G1} to S_{G2}) compensate for the ones on the other side. However, it seems to be the case that the two changes to the medic (halved reload time, but also halved movement speed) counterbalance each other, while the grenadier is much stronger in S_{G2}, and the heavy tank is now so slow that it seems almost unusable.

Step 4: Perform integrated balancing. Manual playtests quickly show that the infantry units are generally too slow, making their use very annoying (this is a subjective impression

[7]https://pypi.python.org/pypi/evoalgos/

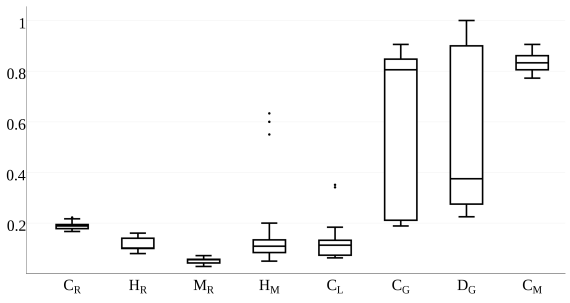


Fig. 4. Boxplot of the parameter values of the best 200 solutions from the automated runs in iteration 1, normalized

and cannot be seen from the raw value). They should have a movement speed of at least 50 to enable strategic maneuvers such as flanking. The heavy tank is also too slow, making it almost useless. Generally, S_{G1} appears favorable over S_{G2}, as it makes better use of the different units. Additionally, the Allies seem to have a slight advantage over the Soviets. Running 100 simulations for S_{G1} confirms that with the Allies winning 85 games.

End: Assess balancing goals. While the third balancing goal (close matches) is technically almost fulfilled, the current best solution S_{G1} does not really enable fair games, and due to the dominance of the light tank and the slowness of the rifleman, we clearly miss the second goal (all units relevant). However, we got some hints on what to change in order to make the game more balanced.

C. Iteration 2

Step 2: Set environment. It is obvious that in the light of the results of iteration 1, we have to readjust our parameter bounds. As a first step, we perform a sensitivity analysis by doubling and halving every parameter value in S_{G1} and running 100 games on each, resulting in 4800 games. The main result of this analysis is that all parameters have noticeable impact on fitness, except from the movement speed of rifle man and medic. This can be explained by their lack of relevance in the meta resulting from the obtained parameter configuration.

In correspondence to our findings in the manual playtests of iteration 1, we fix the movement speeds of the infantry units rifleman, medic, and grenadier to 55, 55, and 50, respectively. Additionally, we fix the reload times (except for the medic) to the values of S_{G1} in order to reduce redundancy. Next, we investigate if some parameter bounds were too tight in the first iteration. A boxplot (fig. 4) of the normalized values of the best 200 solutions from the automated balancing runs reveals that this may be the case for the cost of medic and grenadier as mostly values near 1 were chosen (the damage value for the grenadier in S_{G1} is low as also the mean value is, therefore no correction is needed for D_G). After increasing the upper bounds for the cost of medic and grenadier, we now end up with 17 parameters and slightly adapted parameter bounds.

TABLE IV
SOLUTIONS TAKEN OVER FROM AUTOMATED BALANCING OF ITERATION 2

Parameter	S_{G3}	S_{G4}	S_{G5}
C_H	893	940	1018
H_H	400	422	463

TABLE V
FINAL BALANCING SOLUTION AFTER ITERATION 2

Parameter	Value	Parameter	Value	Parameter	Value
C_R	105	C_M	192	C_G	183
D_R	14	D_M	134	D_G	26
H_R	54	H_M	54	H_G	40
C_L	546	C_H	1018		
D_L	6	D_H	63		
H_L	256	H_H	463		
M_L	65	M_H	40		

Step 3: Perform automated balancing. The second automated balancing step leads to a number of similar best solutions, these cannot be separated into several groups, but they reflect some trade-offs. Especially the cost and health values of the heavy tank are highly correlated, which actually makes sense from a balancing perspective, because the health-per-cost values stays more or less constant. From these solutions, we carry the 3 that are depicted in tab. IV over into the integrated balancing phase, namely the ones with the smallest and the largest values in these two parameters, and an intermediate one.

Step 4: Perform integrated balancing. After multiple playtests with the 3 solutions of tab. IV, we find that unit behavior differences between these solutions are relatively small, hardly noticeable for a novice player. SG_5 feels subjectively a bit better because the heavy tank more resembles the expectation towards heaviness. Performing another set of 100 simulations with SG_5 reveals that now the Soviets have a slight advantage of winning 63 games.

End: Assess balancing goals. The final solution after the second balancing iteration is provided in tab. V. Although the AI wins slightly more often while playing the Soviets, we can consider this solution balanced as the AI is not able to use the medic to its full potential. It keeps the medic near to the infantry units, but takes no precaution to save it from enemy fire, possibly making it an early victim in infantry fights, which in turn weakens the Allies. Manual playtests have shown that proper use of the medic balances this solution well.

VII. CONCLUSION AND OUTLOOK

Balancing is a notable challenge for the game industry that is relevant throughout the entire game development process. Its difficulty increases further whenever many game constituents are created in an automated way or by external actors (e.g., by the community of an existing game). Balancing a full-fledged, commercial game does not yet seem possible with the tools and techniques described in Section II, and manual balancing approaches are very time-consuming and costly, and mostly based on heuristics and rules of thumb (e.g. “doubling and

halving"). Therefore, in this work we have adjusted a process model for manual and automated game balancing aiming to combine the advantages of both approaches while avoiding their pitfalls. This process was adapted and then applied to an existing open source RTS game.

In the original proposal by BEYER ET AL., a clear distinction was made between manual and automated balancing activities. Here, we acknowledge that while automated (optimization) runs are able to identify a number of candidates for good and diverse solutions, the inherent difficulty of formulating a good objective function makes it necessary to combine them with manual playtests for checking if the game is not only fair, but also playable and, most importantly, subjectively entertaining. Additional assessment simulations (or parameter tests) can help to understand the parameter interactions and also find relevant parameters or check the effects of a single change. The result is a truly *integrated* process for game balancing that we presented in Section IV. Summarizing, the most important tools in our toolbox are:

- correlation checks for fitness function justification
- automated balancing via optimization
- doubling and halving of parameters
- manual playtesting
- assessment simulations

Within the guidelines of the overall process, these can be used as appropriate. As possibly subjective human assessment is included here, there is no fully specified algorithm that can solve all problems in one go, balancing is and stays an iterative process. As we acknowledge the importance of human input for this integrated approach, it is difficult to compare to purely algorithmic approaches. However, we do not see this work as gradual improvement but as first (successful) application of an integrated balancing approach to a *real* game with many parameters.

The algorithmic parts of our approach can surely be improved. Grouping the best solutions to clusters can be automated, and the optimization algorithm may be adapted further to the problem, especially concerning noise handling.

Given the inherent redundancy of the balancing problem (most changes of one property of a unit can be counterbalanced by an adequate change in another one), it is no surprise that the problem, seen as an optimization problem, is highly multimodal. Therefore, specific algorithms for this setting should be applied, as well as multi-objective optimization algorithms, which would enable optimizing towards two or more goals at the same time.

REFERENCES

[1] Philipp Beau and Sander Bakkes. "Automated Game Balancing of Asymmetric Video Games." In: *IEEE Conference on Computational Intelligence and Games (CIG)*. 2016.

[2] N. Beume, T. Hein, B. Naujoks, N. Piatkowski, M. Preuss, and S. Wessing. "Intelligent anti-grouping in real-time strategy games". In: *2008 IEEE Symposium On Computational Intelligence and Games*. 2008, pp. 63–70.

[3] Marlene Beyer et al. "An Integrated Process for Game Balancing". In: *Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG 2016)*. 2016.

[4] William Cachia, Antonios Liapis, and Georgios N. Yannakakis. "Multi-Level Evolution of Shooter Levels". In: *AIIDE*. 2015.

[5] S. H. Chang and N. Y. Yang. "DCA: Dynamic Challenging Level Adapter for Real-time Strategy Games". In: *IEEE 15th International Conference on Computational Science and Engineering*. 2012, pp. 30–35.

[6] A. E. Eiben and James E. Smith. *Introduction to Evolutionary Computing*. 2nd. Springer, 2015.

[7] J. Hagelback and S. J. Johansson. "Measuring player experience on runtime dynamic difficulty scaling in an RTS game". In: *IEEE Symposium on Computational Intelligence and Games (CIG)*. 2009, pp. 46–52.

[8] Pier Luca Lanzi, Daniele Loiacono, and Riccardo Stucchi. "Evolving maps for match balancing in first person shooters". In: 2014.

[9] Antonios Liapis, Georgios N. Yannakakis, and Julian Togelius. "Sentient Sketchbook: Computer-Aided Game Level Authoring". In: *8th International Conference on the Foundations of Digital Games*. 2013, pp. 213–220.

[10] Antonios Liapis, Georgios N. Yannakakis, and Julian Togelius. "Towards a Generic Method of Evaluating Game Levels". In: *9th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. 2013, pp. 30–36.

[11] M. Morosan and R. Poli. "Evolving a designer-balanced neural network for Ms PacMan". In: *Computer Science and Electronic Engineering Conference (CEEC)*. 2017, pp. 100–105.

[12] Jacob Kaae Olesen, Georgios N. Yannakakis, and John Hallam. "Real-time challenge balance in an RTS game using rtNEAT". In: *IEEE Symposium On Computational Intelligence and Games (CIG)*. 2008, pp. 87–94.

[13] Edward J Powley, Swen Gaudl, Simon Colton, Mark J Nelson, Rob Saunders, and Michael Cook. "Automated Tweaking of Levels for Casual Creation of Mobile Games". In: *Proceedings of the 2nd Computational Creativity and Games Workshop*. 2015.

[14] Jesse Schell. "Game Mechanics Must be in Balance". In: *The Art of Game Design: A Book of Lenses*. Morgan Kaufmann, Burlington, MA, 2008. Chap. 11, pp. 171–206.

[15] Gillian Smith, Jim Whitehead, and Michael Mateas. "Tanagra: Reactive planning and constraint solving for mixed-initiative level design". In: *IEEE Transactions on Computational Intelligence and AI in Games* 3.3 (2011), pp. 201–215.

[16] Pieter Spronck, Marc Ponsen, Ida Sprinkhuizen-Kuyper, and Eric Postma. "Adaptive game AI with dynamic scripting". In: *Machine Learning* 63.3 (2006), pp. 217–248.

[17] Julian Togelius and Noor Shaker. "The search-based approach". In: *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. Ed. by Noor Shaker, Julian Togelius, and Mark J. Nelson. Springer, Berlin, Germany, 2015, pp. 17–30.

[18] Vanessa Volz, Günter Rudolph, and Boris Naujoks. "Demonstrating the Feasibility of Automatic Game Balancing". In: *Genetic and Evolutionary Computation Conference (GECCO 16)*. ACM Press, New York, NY, 2016, pp. 269–276.

[19] Richard A Watson and Jordan B Pollack. "Coevolutionary dynamics in a minimal substrate". In: *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*. Morgan Kaufmann Publishers Inc. 2001, pp. 702–709.

Applying Commitment to Churn and Remaining Players Lifetime Prediction

Luiz Bernardo Martins Kummer, Julio Cesar Nievola and Emerson Cabrera Paraiso
Graduate Program in Informatics
Pontificia Universidade Catolica do Parana
Rua Imaculada Conceicao, 1155, Curitiba-PR, Brazil
{luiz.kummer, nievola, paraiso}@ppgia.pucpr.br

***Abstract*—In this paper, we present a Machine Learning approach based on commitment to deal with two risky situations over game usage lifecycle: the prediction of churn and players remaining lifetime. These risky situations are gauged by game producers to try to maintain the players motivated, intervening when players tend to leave. The problem is that this information is not trivial to obtain due to the players' motivational usage linked to the many activities available in the game content (e.g., all the possible actions in an MMORPG). To deal with that, we proposed the use of the commitment concept to assign the engagement of a player to a game based on the usage data. The main aspect of the proposed approach is related to the preprocessing step, where measures for the actions related to commitment were computed, generating a tendency degree for each one. We used a dataset from the CIG 2017 Game Data Mining competition which focused on the same risky situations as a comparison baseline. This paper approach overcomes the competition's results for both challenges. All experiments followed the competition rules. These experiments show that looking at tendencies on commitment is an option to gauge players' engagement over time.**

***Index Terms*—game analytic, player behavior, player commitment, churn prediction, remaining lifetime prediction, data mining.**

I. Introduction

The players' engagement to a game changes over time according to what content is available to them. To captivate new players and to motivate the active ones, game producers offer new game content through new games or game upgrades [1]. Artificial intelligence techniques can be applied to the new game content, affecting the gameplay experience of the player [2]. A "clue" to try to identify the degree of experience (motivation in playing) is analyzing the players' generated data when they play (the usage data). This motivation can be represented by "how a player plays" [1]. Approaches to try to measure it are usually done through Data Mining techniques [3]. Usage data can have different granularities, depending on how detailed are the players' activities (in-game actions).

During the usage lifecycle, there are risky and good situations that game producers gauge over time to try to minimize (the risky ones) or to maximize (the good ones) them [1]. This paper focus on two risky situations: the prediction of the remaining players' lifetime (regression) and the churn (classification). Churn is a situation when the player abandons a given game, and the remaining lifetime refers to how long a player will stay playing until he/she churns. These topics present a challenge for prediction because they differ from traditional approaches due to the players' motivational usage [4]. A churn in a given game can differ from a churn of a service because there is no deregistration process associated. The player can stop playing due to several reasons [5][6].

We understand that to better measure how motivated are the players, we need to evaluate as many as possible the different activities that players do playing a game. Related to this idea, commitment is the amount of time played and score obtained [7] by a player during his/her play. We expanded the idea of [7] to other players activities which we understand that also refer to players' motivation.

Our approach was applied over a dataset containing 10,000 players of the MMORPG Blade&Soul[1]. The dataset was made available for the Game Data Mining competition at the CIG 2017. The dataset has labels for churn and lifetime. There are 568,992,669 instances where each one represents an activity done by a player in a distinct time-stamp. The dataset time-span contemplates 24 weeks. During this period, the game changed its business model from a subscription model (pay-to-play) to a free-to-play model. There are 82 different types of activities detailed in the data (e.g. spend money, acquire quest, PvP kill, dismiss guild, etc.).

In this paper, we manually selected 54 out of 82 activities (actions) of the dataset based on the commitment concept [7]. We assume that "if a player likes the game, he/she will spend more time playing and will improve his/her abilities". As an active player is "writing" his/her history in the usage data while playing, we understand that it is better to try to identify the future of this history if we have a clue about it. Therefore, we computed a new attribute to each player's action to measure its tendency and the resultant dataset from this preprocessing has 122 attributes and was used to induce models to deal with both predictions of churn and lifetime. The same rules of the competition were applied to this paper to allow a baseline comparison. As a final result, our proposed strategy overcame the best teams for both challenges.

The paper is organized as follows: next related works are presented. Then, the commitment approach is explained. Following, we present the competition details. The results of some experiments are also shown. Finally, we conclude with

[1]http://www.bladeandsoul.com/en/

978-1-5386-4359-4/18/$31.00 ©2018 IEEE

a discussion on our approach and present possible directions for future work.

II. Related Works

This section is divided into three topics regarding game usage lifecycle modeling, prediction of players' behavior and the approaches presented at the CIG 2017 Game Data Mining competition.

A. Game Usage Lifecycle Modeling

The usage of software was modeled by [8] as the number of active users a software has at a given time-span. This number changes over time due to many causes, such as the software acceptance. In games it is not different, players also have an acceptance stage, but differently from users of "commercial software" (e.g., a text editor), a player has something else, a motivational factor related to entertainment [6].

Most of all games are developed to profit, and it is related to how well the game content keeps players attached. Game producers extract some metrics from usage data to try to identify the players' motivation, such as the MAU (monthly active users), DAU (daily active users), Sticky Factor (DAU/MAU) and revenue [1], but those metrics give the idea of "how many players play", and not "how attached the active players are". One alternative for that was the work of [7], where they proposed a way to measure the commitment of players, given an alternative to the empirical interpretations done by game producers. Their approach could identify risky situations where the classical approach (MAU) could not. The concerns around the usage lifecycle can be divided into two perspectives: one for game producers and other for players.

The game producers aim at identifying when a good or risky situation is happening to make decisions. For example, when the abandon rate is greater than the new players' rate, an upgrade can be applied to motivate the players again with some novelties to know [1]. On the other hand, when a good situation is identified (e.g., a great revenue), the game producer can opt to expand its games to other platforms [9][10]. There are many business models applied to games, where ones have too many players and low revenue and others have the opposite. Sometimes, game producers choose to change its business models, as occurred to the Blade&Soul.

Players have a motivation in continuing playing and it is associated to the content available to them. Some researchers focused on identifying the players' motivational stages over the usage lifecycle. The work of [6] interviewed players from an MMORPG and identified four motivational stages:

- Try: the players have a curiosity about the game and will try it for the first time. If they do not like it, they will abandon it.
- Tasting: after a first good experience, the players start to accumulate profit (e.g., levels, items, friends, quests, etc.).
- Retention: the players already know all the challenges provided to them and start to have a lack of motivation. They continue to play because their friends are playing.
- Abandonment: in this stage, the game content does not please the player and nothing can change his/her mind about leaving the game.

Some researches showed that contacting a player when he/she starts to demonstrate a lack of motivation can change the player's mind [1][11][12].

B. Player Behavior Prediction

Some researchers dealt with the challenge of predicting and identifying players profiles based on usage data associating it with the game usage lifecycle. However, there are also works that identify profiles without doing this association.

In [1], the author was motivated for a "good" problem presented by game producers that have got a great success with their games, "how will the MAU behave?". Some metrics that affect the MAU were identified and then a dynamic system was built based on it. The research final product was a system capable of predicting the MAU's behavior based on changes in current metrics, like the new players' rate and abandonment rate.

The remaining lifetime and churn prediction were object of study for [13] and [14]. In [13], the authors analyzed the login rate changes until players churn. In the case of [14], the measure analyzed was the amount of played time. In both cases, the measures presented a gradual decay until the churn occurrence.

The commitment idea used in this paper was based on the work of Kummer and colleagues [7], where the authors proposed a way to measure the players' commitment to three different degrees (low, average or high). They gauged the changes on the number of players on each commitment degree and proposed a KRI (Key Risk Indicator). They compared the MAU and KRI behaviors during game upgrades of an MMORPG, the proposed metric could identify a risky situation (discontentment associated to upgrade) when the MAU did not.

The works of [15] and [16] studied the aspects related to the identification of players' profiles (different gameplay styles) and which clustering algorithm fits better to that identification. However they have got interesting results, these approaches were not associated with the motivational stages of players during the usage lifecycle.

C. Approaches at CIG 2017 Game Data Mining Competition

This edition of the competition ended in the middle of August 2017. No published papers were found. However, a summary[2] of the competition was presented by its organizers. The competition had got a total of 264 enrolled and started in March 2017 and ended in August 2017.

For the churn prediction challenge, there were 13 teams that delivered a solution (Table I) and for the lifetime prediction challenge were five teams (Table II). It is possible to identify that there were three main approaches applied (Neural network, Trees and Linear models). The ranking process is detailed later in this paper. Unfortunately, no preprocessing

[2]https://cilab.sejong.ac.kr/gdmc2017/index.php/results/

steps were described, therefore it is not possible to compare our approach in this context.

TABLE I
CHURN PREDICTION RANKING

Rank	Team	Techniques
1	YD (Japan)	LSTM+DNN, Extra-Trees Classifier
2	UTU (Finland)	Logistic Regression
3	TripleS (South Korea)	Random Forest
4	TheCowKing (South Korea)	LightGBM (Light Gradient Boosting Machine)
5	goedleio (German)	Feed Forward Neural Network
6	MNDS (South Korea)	Deep Neural Network
7	DTND (South Korea)	Generalized Linear Model
8	IISLABSKKU (South Korea)	Tree Boosting
9	suya (South Korea)	Deep Neural Network
10	YK (South Korea)	Logistic Regression
11	GoAlone (South Korea)	Logistic Regression
12	NoJam (South Korea)	Decision tree
13	Lessang (South Korea)	Deep Neural Network

TABLE II
LIFETIME PREDICTION RANKING

Rank	Team	Techniques
1	YD (Japan)	Ensemble of Conditional Inference Trees (# of Trees = 900)
2	IISLABSKKU (South Korea)	Tree Boosting
3	UTU (Finland)	Linear regression
4	TripleS (South Korea)	Ensemble Three Method
5	DTND (South Korea)	Generalized Linear Model

III. COMMITMENT PROPOSED APPROACH

This section presents our approach to the churn and remaining lifetime prediction. The main contribution of this paper regards the preprocessing step because it allows to construct a training set with a richer information than the original data. The new attributes aim at giving a clue about the future behavior based on observed changes. The idea is using the same set of attributes to deal with both problems.

A. Commitment: the Basic Concept

We advocate that to establish if a player is attached to a game, besides the time spent playing the game, his/her actions that highlight that attachment (the commitment approach [7]) must be also considered. As players evolve while playing the game, they may change their activities [6]. For example, after reaching a certain level, a given player can stop doing daily quests and start to join Player versus Player (PvP) battles. Or start to upgrade his/her equipment when the player's max level is reached. On the other hand, there are actions that may not suggest if a player is attached to the game, like the action of removing an item. Players have a ludic and exploratory behavior, and like to have the capacity of completing difficult challenges, as described by the Flow theory of [17] (Figure 1). The Flow describes some players' characteristics, such as the anxiety derived from a too difficult challenge when the player does not have the required skill or the boredom when the challenge is too easy. The pleasure in playing stays in a balance between the players' ability and the challenge difficulty. It is possible to assimilate changes on players' commitment with the Flow concept, because while a player evolves his/her skills (becoming more committed to the game), he/she may do more difficulty activities ("following the flow"). We assume that the Flow concept applied to this paper can be seen as the growth of occurrence of player's actions together with the beginning of new activities (as the player is becoming more skilled, they may do more of the same or start to do something new).

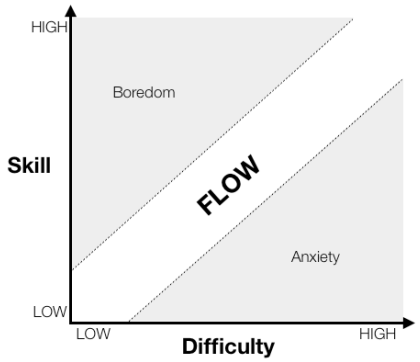


Fig. 1. The flow, extracted from [17]

B. Proposed Method

To apply this method, the candidate game must have usage data with (method assumptions):

- Player identification.
- Time-stamp.
- Player score.
- Action taken.
- Label for Churn and Remaining Lifetime.

The proposed method adds new attributes to the original commitment concept. The idea is to measure all activities related to the players' engagement and then generate new attributes based on them to try to identify a tendency behavior. Players can have different life routines, so it is needed to look individually to each one to detect a change in his/her engagement. Before preprocessing the dataset, it is required to define a time-span to apply the method (e.g., weekly). Then, the following steps for each committed related actions can be applied:

1) To count for each time-span the number of occurrence;
2) To normalize the occurrence number into a range from 0 to 1 for the whole time-span (e.g., all weeks);
3) To compute the mean over the whole time-span;
4) To compute the tendency.

The commitment original measures are not actions [7], they are states regarding obtained score and time spent playing. In our approach, the score is a single attribute regarding the max score obtained in the whole time-span (it represents the player's best performance; e.g., in MMORPGs, the score can be the obtained level, although, there are other attributes that can measure it in more details, such as the occurrence of "level up") and the played time has its mean and tendency values (its granularity may be chosen, like the number of days played).

The normalization step aims at putting "in a same base" players with distinct behavior. A daily player and a weekend

player will be analyzed in the same way through changes in the normalized value. If the original value were used, we can have very distinct values for each player, turning a comparison not fair (e.g., a daily player with 100 completed quests and a weekend player with only 10 may have the same commitment to the game).

The tendency value represents if, at the last time-span, a given action occurrence growth, stayed stable or dropped. The motivation behind this measure consists in finding a "clue" about the future behavior of some action, as it represents the last "feeling" about it considering the historical behavior. The tendency range is from 0 to 1, where a value below 0.5 means a drop of occurrence, a value of 0.5 means stability and a value above 0.5 means a growth of occurrence. It was kept into a positive range to try to avoid wrong computations in the regression models' equations. The tendency is computed as follows: assuming $S_1, ..., S_n$ as the occurrences (normalized) of a player's action for a given time-span, the differences between S_1 and S_2 until S_{n-1} and S_n are summed and then normalized (Equation 1).

$$\text{Tendency} = \frac{(((\sum_{i=1}^{n-1} S_i - S_{i+1}) * -1) + 1)}{2}. \quad (1)$$

where the sum $(\sum_{i=1}^{n-1} S_i - S_{i+1})$ refers to the changes on the occurrence and the others operators besides the sum $(\frac{((\sum)*-1)+1)}{2})$ act only to keep the tendency with the range (0 to 1) and the properties previously described.

To keep this explanation as clear as possible, let us name the original dataset as "basic attributes set" (BAS) and the pre-processed one as "pre-processed attributes set" (PAS). The final product of this preprocessing is the generation of two new attributes for each basic attribute (each commitment related action), which are its mean and tendency values. At this moment, the PAS has only the player identification, the original commitment attributes and the new commitment attributes (mean and tendency values for all commitment related actions). To use PAS to the churn problem, just the class attribute must be added, and to use it to the lifetime problem just the remaining lifetime must be added. Next, a running example will present more details about the preprocessing step.

To better clarify our approach, let us assume the Table III data as our example dataset. The time-span adopted is weekly (eight weeks in total) and the action considered is "Acquire Quest". Note that this player did not acquire quests during all the 8 weeks, only between the 1st and 5th weeks.

The objective of the first step regards counting the number of occurrence of a given action per week. Table IV illustrates the first step result.

The second step consists in transforming the occurrence counts into a range between 0 and 1, where 1 represents the max count identified in the previous step. Table V shows the normalization result. If there are another example with values of 100, 80, 70, 80, 20, 10, 90 and 0, the normalized values will be 1, 0.8, 0.7, 0.8, 0.2, 0.1, 0.9 and 0 respectively. This operation consists in dividing the occurrence value by the max occurrence value (Equation 2).

$$\text{NormalizedValue} = \frac{S_i}{Max(S)}. \quad (2)$$

TABLE III
RUNNING EXAMPLE DATASET

Player_id	Log_name	Date
Player_1	AcquireQuest	14:30 01/01/2018 (week 1)
Player_1	AcquireQuest	15:00 01/01/2018 (week 1)
Player_1	AcquireQuest	20:45 08/01/2018 (week 2)
Player_1	AcquireQuest	20:55 08/01/2018 (week 2)
Player_1	AcquireQuest	21:10 16/01/2018 (week 3)
Player_1	AcquireQuest	09:10 25/01/2018 (week 4)
Player_1	AcquireQuest	09:30 25/01/2018 (week 5)

TABLE IV
OCCURRENCE COUNTING RESULT (FIRST STEP)

Player_id	Log_name	S1	S2	S3	S4	S5	S6	S7	S8
Player_1	AcquireQuest	2	2	1	1	1	0	0	0

TABLE V
NORMALIZATION RESULT (SECOND STEP)

Player_id	Log_name	S1	S2	S3	S4	S5	S6	S7	S8
Player_1	AcquireQuest	1	1	0.5	0.5	0.5	0	0	0

The mean computation step consists of computing for all the eight weeks' normalized values their mean, which was 0.4375 in this case. Next, to obtain the tendency value (last step) the application of Equation 1 is required; the value obtained for the running example was 0. This value means that at the 8th week, the action "AcquireQuest" for "Player_1" tends to drop.

Given the running example, some details about the generated measures are described next. The tendency can be interpreted as follows:

- If a player did not do a given activity in the first week and did it in the last week, then a value greater than 0.5 will be obtained. This shows the idea of having a new behavior (doing something new). If the tendency value is 1, it means that the last player behavior has the highest occurrence.
- If a player did not do a given activity in the first and last weeks, then a value of 0.5 will be obtained. This shows the idea of a stable behavior. The same occurs if a player does an activity in the first and last weeks with the same occurrence.
- If a player did a given activity in the first week and did not do it in the last week, then a value lower than 0.5 will be obtained. This shows the idea of having a new behavior (stop doing something, like the running example). The same occurs when a player does an activity in the last week with less occurrence than in the first week.

The mean measure can be interpreted with the following possibilities:

- A value near 1 means that the player does a certain activity with a stable behavior.
- A value near 0.5 means that the player changed its behavior (but we do not know when it happened; the tendency idea acts here).
- A value near 0 means that the player rarely does an activity.
- A value of 0 means that the player does not do a certain activity.

We proposed to use the tendency value to give an extra information about the player behavior, because we understand that using only the mean value does not fit well, as it may hide some information, like abrupt changes on behavior.

The mean and tendency values analyzed together can have a richer information, such as:

- When the mean value is near 0 and the tendency is near 1, it means that the player started to do something new in the last week.
- When the mean value is near 1 and tendency is near 0, it means that the player stopped abruptly to do something.
- When both values are near 1, it means that the player has a stable behavior and the last week have one of the highest occurrences.
- When both values are near 0, it means that the player tried a new activity which was discontinued.

After preprocessing, the Data Mining step may start.

IV. The CIG 2017 Game Data Mining Competition

Blade&Soul is a famous MMORPG in South Korea and was released in June 2012. The dataset contains 568,992,669 instances (about 100GB) and is divided in three subsets: one for training and two for test, where the test sets do not have the labels related to churn (Yes or No) and the lifetime (e.g., 30 days) (the lifetime granularity defined to the competition is daily). Each instance represents one of 82 possible players' actions in-game, such as getting an item, spending money or winning a PvP battle.

The proposed challenges consist in measuring the loyalty of players to the game in two perspectives, if the player will abandon the game and how long he/she will continue to play. The dataset time-span contemplates a total of 24 weeks, but the data are not continuous in time. There is a distinct period for each subset (training and test sets; eight weeks for each). In December 2016, the game changed its business model from subscription to free-to-play. It gave to the competition an extra challenge because the behavior of test set 2 tended to change. A summary of each subset is described in Table VI. It is important to highlight that the labels from the test sets were available only after the end of the competition. During the competition time, only the training set had values for churn and lifetime.

In order to rank the competition results, the organizers used for the churn challenge (classification problem) the F1 score and for the prediction of the remaining lifetime (regression) the Root Mean Squared Logarithmic Error (RMSLE) (3).

$$\text{RMSLE} = \sqrt{\frac{\sum_{i=1}^{n}(\log(p_i+1)-\log(a_i+1))^2}{n}}. \quad (3)$$

where n is the total number of observations, p_i is the predicted value and a_i is the actual response value (the real value).

The churn definition was also provided by the organizers. As in games like Blade&Soul, players do not simply do a deregistration, they simply stop playing. It is not much precise to define when a player churned because he/she can just stop for a period and then came back again. The organizers took it into account and proposed the following model. The churn occurs based on three-time slices, the first one with eight weeks, the second one with three weeks and the last one with five weeks. The slices are continuous (the end of the first corresponds to the start of the second and so on). If a player plays in the first slice and does not play in the third, then he/she churned. The second slice can be considered a period of no data. If the player played just one day in the third period, he/she did not churn.

The 10,000 players presented in the dataset were selected by the organizers based on the following criteria:

- They are not bots (malicious users) (examples in [18]).
- They are not "light users" (players who are not loyal and not profitable).

It is interesting to notice that the game producer focuses only on the "best" players, leaving apart the "not so promising players". As bots present a distinct behavior [19], their exclusion helps in removing some noise from the dataset.

V. Experiments

As the Blade&Soul dataset is divided into weeks, we assume the weekly granularity to the proposed method. Table XI shows all the attributes presented in the usage data and which ones were selected based on commitment. Besides the 54 attributes chosen (from the total of 82) we added more 8 (totalizing 62), which are:

- Player max level during the eight weeks: this attribute has a maximum value of 50. At the beginning of the game, a new player will evolve this level.
- Player max mastery level during the eight weeks: the mastery level starts to grow when the player reaches the maximum player level (50).
- The average session time.
- The number of days played.
- Splitting the "PartyBattleResult(PC)" into "WinPartyBattleResult(PC)" and "LosePartyBattleResult(PC)".
- Splitting the "PartyBattleEnd(Team)" into "WinPartyBattleEnd(Team)" and "LosePartyBattleEnd(Team)".
- Splitting the "DuelEnd(PC)" into "WinDuelEnd(PC)" and "LoseDuelEnd(PC)".
- Splitting the "DuelEnd(Team)" into "WinDuelEnd(Team)" and "LoseDuelEnd(Team)".

TABLE VI
SUBSETS SUMMARY

Subset	Number of Players	Business Model	Start Date	End Date	# Churn = No	# Churn = Yes
Training	4000	Subscription	2016-03-16	2016-05-11	2800	1200
Test 1	3000	Subscription	2016-07-27	2016-09-21	2100	900
Test 2	3000	Free-to-play	2016-12-14	2017-02-28	2100	900

The commitment original measures [7] are contemplated into the following attributes: average session time, number of days played, player max level and player max mastery level. After the preprocessing step, the PAS (preprocessed attributes set) contains 122 attributes, where an amount of 120 was obtained processing 60 attributes (mean and tendency) and two referencing the players' levels (inside the attributes chosen, there is the "Level Up" action, which describes into a more detailed way the evolution of the players' level). This process was applied to the Test datasets too, the total of 568,992,669 original instances from all datasets was pre-processed into 10,000 instances (4,000 for training, 3,000 for Test 1 and 3,000 for Test 2; these datasets are available at https://www.ppgia.pucpr.br/(tilde)paraiso/Projects/GameAnalytic/DataBases/BladeAndSoul/).

The proposed challenges consist in better predicting the correct values for churn and lifetime (Test dataset 1 and Test dataset 2) according to the Training dataset. As we are following the competition rules, we looked only to the labels in the Training set, although the Test datasets labels became available after the competition.

We tested four different algorithms for the churn identification, which were: C4.5, RepTREE, MLP and SVM. All with the default configuration. The algorithms tested to the lifetime prediction were: M5P, RepTREE and MLP. All with the default configuration. The classification and regression results are presented in the next section.

VI. RESULTS AND ANALYSIS

Before applying the classification and regression models over the test sets, we had to opt which one is the chosen one for each challenge based on the training results. Although we identified a statistical difference between the classifiers' predictions (comparing each instance prediction; test t with $p < 0.05$), as the classification challenge is evaluated by the F1-score, we chose the algorithm with the best F1-score in the training, which was the C4.5. Table VII shows all algorithms results for the classification (10-fold cross-validation), also including the test scores.

TABLE VII
CLASSIFICATION: TRAINING AND TEST RESULTS

	Training		Test (F1-score)		
Algorithm	**Accuracy**	**F1-score**	**Test 1**	**Test 2**	**Test mean**
C4.5	75.9	0.757	0.712	0.738	0.725
RepTREE	75.675	0.751	0.721	0.730	0.7255
MLP	71.95	0.715	0.648	0.655	0.6515
SVM	76.925	0.753	0.721	0.726	0.7235

Comparing the C4.5 result with the competition result we get the rank presented in Table VIII (only the first five teams are shown).

TABLE VIII
CHURN FINAL RESULT COMPARISON

Rank	Team	Test1 score	Test2 score	Total score
1	This approach	0.712	0.738	0.725
2	YD (Japan)	0.61008	0.63326	0.62145
3	UTU (Finland)	0.60326	0.60370	0.60348
4	TripleS (South Korea)	0.57968	0.62459	0.60130
5	TheCowKing (South Korea)	0.59370	0.60718	0.60036

The C4.5 generated tree had a max deep of six levels (21 nodes) and its attributes near the root were the mean of the number of days played and the player's max mastery level. We compared the performance of this approach for Test 1 and Test 2 with the other approaches presented in Table VIII using a t test with $p < 0.05$ and statistical differences were found.

The same was done for the regression problem, but here the evaluation is based on the RMSLE (Equation 3). Table IX shows the training (10-fold cross-validation) and test results. It is important to report that for M5P and MLP, it was not possible to compute their RMSLE values for training and test datasets. As they predicted negative values, an error is generated in Equation 3, because there are no values for a negative log. We opt to not apply a normalization in this case because we could not find a reason to use a prediction of a remaining lifetime with negative values (it looks wrong). We manually applied the generated regression equations for M5P and MLP and confirmed that in some cases, negative values are generated. The algorithm chosen for the regression problem was the RepTREE.

TABLE IX
REGRESSION: TRAINING AND TEST RESULTS

	Training	Test (RMSLE)		
Algorithm	**RMSLE**	**Test 1**	**Test 2**	**Test mean**
M5P	Computation error	-	-	-
RepTREE	0.4426	0.4722	0.4413	0.4568
MLP	Computation error	-	-	-

The final rank for the lifetime prediction is presented in Table X.

The RepTREE generated tree had a max deep of seven levels (47 nodes) and its attributes near the root were the same as for the classification problem. We compared the performance of our approach for Test 1 and Test 2 with the other approaches presented in Table X using a t test with

TABLE X
LIFETIME FINAL RESULT COMPARISON

Rank	Team	Test1 score	Test2 score	Total score
1	This approach	0.4722	0.4413	0.4568
2	YD (Japan)	0.883248	0.616499	0.726151
3	IISSLABSKKU (South Korea)	1.034321	0.679214	0.819972
4	UTU (Finland)	0.927712	0.898471	0.912857
5	TripleS (South Korea)	0.958308	0.891106	0.923486
6	DTND (South Korea)	1.032688	0.930417	0.978888

$p < 0.05$ and statistical differences were found (as occurred to the classification).

VII. CONCLUSION AND FUTURE WORKS

In this paper, we present a commitment based approach to churn and remaining players lifetime prediction. We applied the approach to the CIG 2017 Game Data Mining Competition data. It is interesting to notice that even if the game changed its business model (Table VI), our approach had got similar results to both Test datasets, whereas the other teams presented a certain variance between them (especially in the lifetime prediction; Table X). It means that the commitment approach looks to have a certain generic approach over different business models, showing that similar players' behavior can exist between them. The fact that this paper approach got the first place in the two challenges foments that commitment is an option to gauge players' engagement over time. Experiments with new data may confirm, or not, these first conclusions.

For each attribute accepted in Table XI, the preprocessing step produced two new attributes, the mean and the tendency. Their combination was valid, because the generated decision trees (C4.5 and RepTREE) used both attributes into the decision nodes, emphasizing their interpretations contained in the Commitment Proposed Approach section (especially in the running example observations). It is also interesting to notice that the attributes near the root for both models were very similar, focusing on the time spent playing and the score obtained (player's level), giving support to the commitment definition of [7]. The generated new attributes were used in deeper levels of the decision trees, acting as specifiers inside the time spent playing and obtained score universe. Although not all new attributes were used by the classifiers, they may be used to deal with other prediction challenges.

We also note that all the classification and regression algorithms tested (except the ones with computation errors) outperformed the competition results. It foments the fact that the preprocessing step is the key aspect of this work. The generation of attributes regarding the tendency of each action gave an extra information to the models allowing a better performance.

As future works, we intend to apply this approach to other games and compare the results (considering other game genres too). We also want to apply an automatic feature selection algorithm, and check if the attributes chosen are the same as selected in this paper. It can give more robustness to this approach.

ACKNOWLEDGMENT

We would like to thank NCSOFT for turning the dataset of Blade&Soul available and to CAPES-Brazil (Coordenação de Aperfeicoamento de Pessoal de Nivel Superior) for its support in this research.

REFERENCES

[1] T. H. Speller III, "The business and dynamics of free-to-play social-casual game apps," Ph.D. dissertation, Massachusetts Institute of Technology, 2012.

[2] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[3] L. Galway, D. Charles, and M. Black, "Machine learning in digital games: a survey," *Artificial Intelligence Review*, vol. 29, no. 2, pp. 123–161, 2008.

[4] M. C. Mozer, R. Wolniewicz, D. B. Grimes, E. Johnson, and H. Kaushansky, "Predicting subscriber dissatisfaction and improving retention in the wireless telecommunications industry," *IEEE Transactions on neural networks*, vol. 11, no. 3, pp. 690–696, 2000.

[5] D. Cook. (2007) The circle of life: An analysis of the game product lifecycle. [Online]. Available: \texttt{https://www.gamasutra.com/view/feature/129880/the_circle_of_life_an_analysis_of_.php}

[6] L. Zhu, Y. Li, and G. Zhao, "Exploring the online-game life cycle stages," in *E-Business and E-Government (ICEE), 2010 International Conference on*. IEEE, 2010, pp. 2436–2438.

[7] L. B. M. Kummer, J. C. Nievola, and E. C. Paraiso, "A key risk indicator for the game usage lifecycle," *AAAI Publications, The Thirtieth International Flairs Conference (FLAIRS)*, pp. 394–399, 2017.

[8] G. A. Moore, "Crossing the chasm, 1991," *HarperBusiness, New York*, 1995.

[9] S. Ludgate. (2011) Mmo chat: Scott hartsman. [Online]. Available: \texttt{http://www.gamasutra.com/blogs/SimonLudgate/20110707/89762/MMO_Chat_Scott_Hartsman.php}

[10] B. Sheffield and L. Alexander. (2008) Focus on korea: T3's kim talks hitmaking with audition. [Online]. Available: \texttt{https://www.gamasutra.com/view/news/107795/}

[11] C. S. Müntner, "The peculiar problems of the gaming industry: customer retention in mmoprgs," Ph.D. dissertation, Reykjavik University, 2017.

[12] V. Kumar and D. Shah, "Building and sustaining profitable customer loyalty for the 21st century," *Journal of retailing*, vol. 80, no. 4, pp. 317–329, 2004.

[13] E. G. Castro and M. S. Tsuzuki, "Churn prediction in online games using players' login records: A frequency analysis approach," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 7, no. 3, pp. 255–265, 2015.

[14] P.-Y. Tarng, K.-T. Chen, and P. Huang, "On prophesying online gamer departure," in *Proceedings of the 8th Annual Workshop on Network and Systems Support for Games*. IEEE Press, 2009, p. 16.

[15] A. Drachen, R. Sifa, C. Bauckhage, and C. Thurau, "Guns, swords and data: Clustering of player behavior in computer games in the wild," in *Computational Intelligence and Games (CIG), 2012 IEEE Conference on*. IEEE, 2012, pp. 163–170.

[16] A. Drachen, C. Thurau, R. Sifa, and C. Bauckhage, "A comparison of methods for player clustering via behavioral telemetry," *arXiv preprint arXiv:1407.3950*, 2014.

[17] J. Nakamura and M. Csikszentmihalyi, "Flow theory and research," *Handbook of positive psychology*, pp. 195–206, 2009.

[18] H. Kwon, A. Mohaisen, J. Woo, Y. Kim, E. Lee, and H. K. Kim, "Crime scene reconstruction: Online gold farming network analysis," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 3, pp. 544–556, 2017.

[19] A. R. Kang, J. Woo, J. Park, and H. K. Kim, "Online game bot detection based on party-play log analysis," *Computers & Mathematics with Applications*, vol. 65, no. 9, pp. 1384–1395, 2013.

TABLE XI
SUMMARY OF BLADE&SOUL ATTRIBUTES

Log_id	Log_name	Retained	Motivation
1003	EnterWorld		Already explored in the session time and played days attributes
1004	LeaveWorld		Already explored in the session time and played days attributes
1005	EnterZone		Trivial action not relevant to Commitment
1006	LeaveZone		Trivial action not relevant to Commitment
1010	Teleport		Trivial action not relevant to Commitment
1012	DeletePC		As a player can churn without deleting his/her character, we discarded this attribute
1013	PcLevelUp	X	Related to Commitment
1016	GetExperience		Present in other attributes, such as the PcLevelUp
1017	GetMoney	X	Related to Commitment
1018	SpendMoney	X	Related to Commitment
1022	GetItem	X	Related to Commitment
1023	LoseItem	X	Related to Commitment
1101	InviteParty	X	Related to Commitment
1102	JoinParty	X	Related to Commitment
1103	RefuseParty	X	Related to Commitment
1104	DismissParty	X	Related to Commitment
1105	KickPartyMember	X	Related to Commitment
1201	Exhaustion	X	Related to Commitment
1202	Die	X	Related to Commitment
1203	Resurrect		If a player dies, he/she will resurrect. So, we disconsidered this attribute
1208	KillNPC	X	Related to Commitment
1209	KillPC	X	Related to Commitment
1404	DuelEnd(PC)	X	Related to Commitment
1406	DuelEnd(Team)	X	Related to Commitment
1407	MoveToArena		Trivial action and not relevant to Commitment
1422	PartyBattleEnd(Team)	X	Related to Commitment
1424	PartyBattleResult(PC)	X	Related to Commitment
1425	OccupyBase		This kind of activity is not usual, so we discarded
2001	LootItem	X	Related to Commitment
2002	UseItem	X	Related to Commitment
2004	DestroyItem	X	Related to Commitment
2006	getLootMoney	X	Related to Commitment
2011	PartyAuctionStart		This kind of activity is not usual, so we discarded
2013	BidPartyAuction		This kind of activity is not usual, so we discarded
2014	PartyAuctionSuccess		This kind of activity is not usual, so we discarded
2016	DistributeAuctionMoney		This kind of activity is not usual, so we discarded
2102	UnEquipItem		Trivial action not relevant to Commitment
2103	SaveEquipInfo		Trivial action not relevant to Commitment
2105	RevealItem	X	Related to Commitment
2106	ConsumeGemByReveal		This kind of activity is not usual, so we discarded
2109	GetItemByDecomposition	X	Related to Commitment
2112	ExpandWarehouse		This kind of activity is not usual, so we discarded
2113	RepairItem	X	Related to Commitment
2121	GrowUpItem	X	Related to Commitment
2126	ResultOfTransform		Trivial action not relevant to Commitment
2127	ExceedItemLimit		Trivial action not relevant to Commitment
2141	ChangeItemLook	X	It is related to the commitment idea
2145	FeedingResult		Trivial action not relevant to Commitment
2201	TradeGiveItem	X	Related to Commitment
2202	TradeGetItem	X	Related to Commitment
2204	SellItem	X	Related to Commitment
2205	BuyMyItem	X	Related to Commitment
2206	TradeGiveMoney	X	Related to Commitment
2207	TradeGetMoney	X	Related to Commitment
2209	GetItemFromNPC	X	Related to Commitment
2221	DepositItem	X	Related to Commitment
2222	RetriveItem	X	Related to Commitment
2301	PutMainAuction		This kind of activity is not usual, so we discarded
2307	BuyItemNowMainAuction		This kind of activity is not usual, so we discarded
2405	UseGatheringItem		Trivial action not relevant to Commitment
2407	GetGatheringItem		Trivial action not relevant to Commitment
2503	ExpireEventItem		Trivial action not relevant to Commitment
4001	AcquireSkill	X	Related to Commitment
4002	SkillLevelUp	X	Related to Commitment
4006	LearnTraining	X	Related to Commitment
5001	AcquireQuest	X	Related to Commitment
5004	CompleteQuest	X	Related to Commitment
5005	DisposeQuest	X	Related to Commitment
5006	GetQuestItem		Trivial action not relevant to Commitment
5008	GetQuestSkill		Trivial action not relevant to Commitment
5010	GetChallengeTodayItem	X	Related to Commitment
5011	CompleteChallengeToday	X	Related to Commitment
5014	GetChallengeWeekItem	X	Related to Commitment
5015	CompleteChallengeWeek	X	Related to Commitment
6001	CreateGuild	X	Related to Commitment
6002	DestoryGuild	X	Related to Commitment
6003	GuildLevelUp	X	Related to Commitment
6004	InviteGuild	X	Related to Commitment
6005	JoinGuild	X	Related to Commitment
6008	RefuseGuildInvite	X	Related to Commitment
6009	DissmissGuild	X	Related to Commitment
6010	KickGuildMember	X	Related to Commitment

Neuroevolution for RTS Micro

Aavaas Gajurel, Sushil J Louis, Daniel J Méndez and Siming Liu
Department of Computer Science and Engineering, University of Nevada Reno
Reno, Nevada
avs@nevada.unr.edu, sushil@cse.unr.edu, dmendez@nevada.unr.edu, simingl@unr.edu

***Abstract*—This paper uses neuroevolution of augmenting topologies to evolve control tactics for groups of units in real-time strategy games. In such games, players build economies to generate armies composed of multiple types of units with different attack and movement characteristics to combat each other. This paper evolves neural networks to control movement and attack commands, also called micro, for a group of ranged units skirmishing with a group of melee units. Our results show that neuroevolution of augmenting topologies can effectively generate neural networks capable of good micro for our ranged units against a group of hand-coded melee units. The evolved neural networks lead to kiting behavior for the ranged units which is a common tactic used by professional players in ranged versus melee skirmishes in popular real-time strategy games like Starcraft. The evolved neural networks also generalized well to other starting positions and numbers of units. We believe these results indicate the potential of neuroevolution for generating effective micro in real-time strategy games.**

***Index Terms*—neural networks, evolution, NEAT, RTS micro**

I. Introduction

Real Time Strategy (RTS) games are a genre of multi-player video games where players take actions concurrently and the underlying game world dynamically changes over time. The overarching objective of the the game is to establish a position capable of defending against and destroying opponents. Actions in the game can be largely divided into two modes: "macro" and "micro". Macro management relates to long term strategic decisions and is concerned with resource gathering, spending those resources on research, deciding on the type and number of units to build, and in building those units. Micro management is concerned with quick and short term tactical control of units usually during a skirmish between a group of friendly units against an opponents group. RTS game environments are a partially observable and imperfect information environment due to a restricted view through the camera on a part of the whole map and a "fog of war" which hides information form parts that have not been explored. Players have to control numbers of units ranging from tens to hundreds while simultaneously moving the camera around, deciding which units or unit factories to build, selecting units, scouting, and exploring. The state space of typical RTS games like Starcraft is estimated to be more than $10^{50^{36000}}$ using a conservative branching factor of 10^{50} for each frame in a 25 minute game [1]. Consequently, RTS games provide a challenging platform for testing machine learning approaches [2].

This paper focuses on generating artificial agents capable of good micro control in RTS games. Micro requires quick decision making and fast successive actions to control both movement and attack commands for units in a group. There are multiple types of units, each with its own advantages and disadvantages. Each unit has unique, well-defined characteristics regarding its capabilities, like weaponry, range, speed, maneuverability and others. Good micro can be a deciding factor in a skirmish between two groups with similar characteristics and the player has to consider the attributes of both friendly and enemy units to choose an effective tactic for the particular scenario. The complexity of the different ways in which any unit group can be controlled is as a result challenging, particularly since directives have to be provided in quick reactive time-frames.

RTS games have been used as an environment for AI research and various approaches towards automation of different aspects of RTS game playing have been explored [1]. Approaches like reinforcement learning, scripting, and search, among others, have been used with the end goal of creating a fully automated, human-comparable RTS player [3]. Previous work has explored using Genetic Algorithms (GA) to search for an optimal combination of parameters, which are then used in Potential Fields (PF) and Influence Maps (IM) equations to control the tactical actions of skirmishing units [4]. Our research builds on this previous work in RTS game AI, but takes a different approach. Rather then having a set of parameterized control algorithms, or potential fields, for controlling movement, we explore the feasibility of evolving a neural network to perform good micro. In particular, we explore using Neuro-Evolution of Augmented Topologies (NEAT) [5] to evolve neural networks to effectively and autonomously control units for skirmishes in RTS games.

NEAT evolves both the structure and connection weights of a neural network by utilizing genetic algorithm principles [6] and applying them to a population whose chromosomes represent different instances of networks being explored. Thus, the NEAT approach encodes both the structure and weights of a neural network that tries solve a problem. NEAT incorporates historical markings, speciation and starts complexification from a minimal network. There is good empirical evidence that NEAT can evolve robust solutions for non-trivial problems [7].

We feed the evolving network with the relative positions of all units in the arena along with the unit's internal state as inputs. There are three outputs. Two specify a relative 2D position $(\delta x, \delta y)$ to move towards and third represents a binary value that determines if we move or attack.

Preliminary results, on an underlying RTS-physics imple-

978-1-5386-4359-4/18/$31.00 ©2018 IEEE

menting simulation, show that NEAT can evolve networks for micro control of ranged units against a group of melee units. The evolved network generated kiting behaviour for ranged units (copied from vultures in Starcraft) which allowed five vultures to eliminate twenty-five zealots (a strong melee unit) without suffering any damage. We evolved our network on ten different starting configurations differing in the initial positions and numbers of zealots, and tested the evolved network on configurations with varying numbers of zealots from one to thirty. Our results indicate that evolved networks generalize well to different starting configuration and varying numbers of vultures. We then moved to the recently released Starcraft II (SCII) game API [8] and were able to show that NEAT can evolve good micro on a simple, flat Starcraft II map with no obstacles. Although the networks evolved in SCII are not as effective as those that evolve in our simulation, when pitting 5 hellions against 25 zealots, hellions learn to kite and can on average destroy close to a majority of zealots. [1] As before, the NEAT networks generalized to different numbers of zealots and to different starting locations. We believe our method of network representation can be extended to incorporate new inputs to be applicable to more complex micro scenarios. In addition, we believe that we can significantly improve NEAT evolved micro in SCII with more computational resources.

The remainder of this paper is organized as follows. Section II describes previous approaches related to our current work, Section III describes the neural network representation, evolution configurations and NEAT setup. In Section IV, we describe our generalization results and lastly in Section V we draw our conclusions and explain possible future approaches.

II. Related Work

Significant work has been done over the years in the field of designing effective RTS AI players using different techniques [1]. Buckland *et al.* described a rule based approach in his book [9] and Rabin and Steve [10] explained scripted agents which is a general approach used by bots that play in starcraft AI ladder matches. Weber and Mateas [11] explored using data mining on gameplay logs to predict the strategy of an opponent. A tree based search approach was used by Churchill, Saffidine and Buro who utilized transition tables to generate trees of actions and performed alpha beta pruning to create agents for 8 vs 8 unit skirmish [12]. Others have also tried reinforcement Learning: Wender and Watson [13] used Q learning and Sarsa to develop a fight or retreat agent. Shantia, Begue and Wiering [14] applied reinforcement learning on neural networks by using neural-fitted and online versions of the Sarsa Algorithm where they implemented a state space representation similar to [13]. Vinyals *et al.* [15] applied deep reinforcement learning in a Starcraft II environment to train neural networks using gameplay data from expert players. Their representation used raw image features corresponding to the game display called feature maps and they provide baseline results for convolutional, long-term short-term memory and random policy based agents.

Potential Fields (PFs), which has been widely used for robot navigation and obstacle avoidance [21][22] [16] have also been used for micro. Hagelback and Johansson [17] presented a multi-agent potential field based bot architecture for the RTS game ORTS [18] which incorporated PFs into their unit AI. More recent work has focused on combining PFs with Influence Maps (IM) to represent unit and terrain information. In this context, an influence map is a grid superimposed on the virtual world where each cell is assigned a value by an IM function, which is used by an AI to determine desired actions [19] [4]. Coevolution was used by Avery and Louis in [20] to develop micro behaviours by coevolving influence maps for team tactics and in [21] where they cooevolved influence map trees(IMT) and show that evolved IMTs displayed similar behaviours to hand coded strategies.

NEAT has been applied to dynamic control tasks like double pole balancing without velocity information [5] where it could evolve a robust control policy. It has also been applied to evolving walking gaits for virtual creatures [22] and steering control for driving agents [23] [24]. NEAT has also been applied to evolve video game playing agents for games like Ms. Pac-Man [25] and Tetris [26] and has been shown to be applicable to general Atari gameplaying [7]. Board games like 2048 [27] and Go [28] have also been shown to be within reach.

NEAT and its realtime variant rtNEAT have been used to tackle different aspects of RTS games. Olesen *et al.* [29] used NEAT and rtNEAT to control the macro aspects of the game to match the difficulty of the opponent. Gabriel *et al.* [30] used rtNEAT to evolve a multi-agent system for Brood war agents based on ontological templates, where they show that their hierarchical method could be used to evolve good micromanagement tactics. NEAT for RTS micro control was applied in [31] where the authors approached the problem by having a neural network control a combat unit's fight or flee decision, based on various entity properties like weapon cooldown, remaining health, weapon range, enemy weapon range, number of allies in range and number of enemies in range. They used NEAT and rtNEAT and had the fight or flee logic hard-coded for the network to activate; which differs from the approach in this paper where we are directly trying to control unit movement based on the position of friendly and enemy units around the entity being controlled, without further structure.

III. Methodology

There are different aspects of micro game-play that can be controlled for an entity, such as movement, whether to attack, when to flee, and other such unit specific actions. Controlling all aspects of a micro engagement is therefore a complex endeavor. In this research, we focus on entity movement and firing control. Movement can be further classified into long and short range, based on the timescale within which the action must execute. Longer routes pertain to long distance

[1] Like vultures in Starcraft, hellions are also relatively fragile, longer ranged, and fast Starcraft II units.

movement of units, say from a player's base to an enemy's base, while short duration movement, like kiting, are finer tactical movements done in shorter periods of time. Kiting is a strategy that is used by speedy ranged units against slower melee units, where the ranged units fire, run out of range, turn back, fire, and run back out of range again and again avoiding damage to themselves while damaging the enemy. We are trying to evolve networks which can perform similar tactics for ranged units against melee units.

We next describe the NEAT evolutionary algorithm, the neural network representation for NEAT, and the experimental setup used for evolution.

A. Neuro-Evolution of Augmenting Topologies

NEAT is a robust algorithm for evolving neural networks based on genetic algorithm principles. NEAT attributes it's robustness to three aspects, specifically that it starts complexifing from minimal structure, that it leverages speciation and its use of historical markings in the genome for crossover and speciation [5]. NEAT allows for continuous complexification by allowing mating together with mutation to fully modify the resulting network and a number of different kinds of mutation are used [5].

NEAT evolves a neural network from inputs and outputs specified by the problem domain. We specify the inputs and outputs in more detail below.

Our neural network inputs can be divided into two classes according to the type of information they represent. The first class deals with spatial information and describes the relative position of all entities on the map. In order to be able to represent units consistently and uniformly, regardless of the number, we followed an approach whereby we divide the visible world into regions relative to the current position of the unit being controlled. Figure 1 shows the spatial information being fed into the neural network. In our representation, the

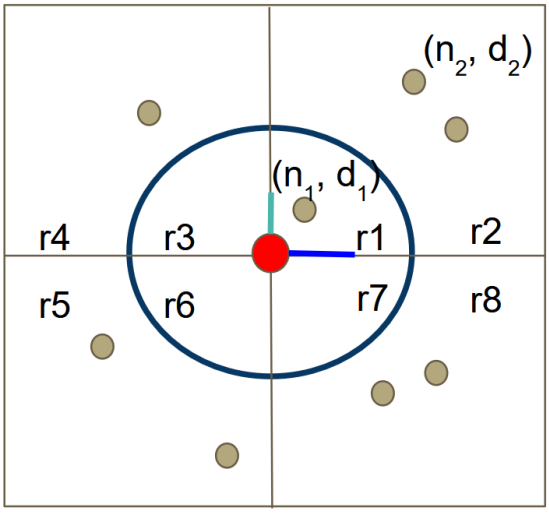


Fig. 1. Input and Output Representation

world around the entity is divided into 4 quadrants with the entity at its center. The four quadrants are further divided into eight regions separated by the attack range of the unit as shown by the labels R1 to R8 in Figure 1. Each region then corresponds to four inputs in the network:

1) the number of enemy units
2) average distance of enemy units
3) number of friendly units and
4) average distance of friendly units in the region

Next, we have four inputs indicating map boundaries. These inputs provide distance from the entity to north, south, east and west boundaries correspondingly. The second class of inputs, feeds the entity with some of the entity's own internal state. The internal features that we have considered are

1) the current health
2) weapon cool-down
3) current fire or move state and
4) a recurrent input which is the previous attack/move output from the network

Thus, in total the neural network that controls the movement of friendly units in our environment has 40 inputs, 16 that are used to represent the position of all friendly units, 16 that are used to represent the position of enemy units, 4 boundary sensors and 4 inputs for the internal state as shown in table I. The representation is constructed such that it can capture essential information from different map configurations and number of entities, without having to vary the number of inputs in the network. Once computed, all inputs are scaled between 0 to 1 by representing each value as percentage of a maximum possible value for that input.

TABLE I
NEURAL NETWORK INPUTS

id	Type
1-8	enemy avg position per region
9-16	friendly avg position per region
17-24	enemy units per region
25-32	friendly units per region
33-36	boundary sensors for 4 directions
37	self cooldown
38	self hitpoint
39	current attack/move state
40	previous attack/move state

1) Output representation: The output is represented by two scalars representing a desired δx and δy coordinate displacement, and one boolean for whether a unit should fire or move at that instant. δx and δy displacement output are scalar values from 0 to 1 from which we subtract 0.5 and then scale them to go the coordinate position relative to the unit's current position. This allows the output to represent any coordinate around the entity in the region corresponding to the scaling factor. The outputs are then fed into the movement mechanism of the simulation or SCII in order to generate movement. The third output is a move or attack command which is a Boolean signal. If the output is greater than 0.5, the entity has to focus on attack and if the output is lower, the entity stops attacking and begins moving to the position

signalled by δx and δy displacement output. To attack, the entity focuses on the nearest opponent in range and fires its main weapon if the weapon is not in cooldown.

B. Experimental setup

Although our physics-simulation used for preliminary results and for experimentation to explore input and output representations runs fast, the simulated physics and entity properties cannot be easily made identical to SCII. This means that micro evolved in our simulation may not transfer well to SCII. In addition, there are differences in the properties of vultures in our simulation, vultures in Starcraft Brood Wars, and hellions in SCII. However, our simulation runs much faster and we can experimentally try multiple representations, input configurations, and NEAT parameters far more quickly than when using the SCII API. We can then start long evolutionary runs within the SCII environment with more confidence.

For our experiments, we choose the zealot as a representative melee unit and either the vulture or hellion as a representative ranged unit. More specifically, for our simulation environment, we copied zealot and vulture properties from the Starcraft BW API [32]. When running in SCII, zealots and hellions use SCII properties. Vultures/hellions and zealots deal comparable damage in each attack but have different attack range and movement speeds. Table II shows the properties of the units considered in this paper. Hellions and vultures, when micro'd well can be strong against zealots because of their greater speed and attack range, which makes it possible for a small number of vultures/hellions to kite a bigger group of zealots to death. We expect our approach to evolve good tactical control that can exploit this strength of vultures/hellions against zealots.

TABLE II
PROPERTIES OF VULTURES, ZEALOTS AND HELLIONS

Property	Vulture	Hellion	Zealot
Hit-points	80	90	100
Damage	20	13	16
Attack Range	5	5	0.1
Speed	4.96	5.95	3.15
Cool-down	1.26	1.78	0.857

NEAT evolves the network across generations based on the fitness of the network. To evaluate the fitness of the neural network, we used two different environments: the Starcraft II game and our own simulation of the Starcraft environment which is tailored to capture the micro combat aspects of Starcraft and can be run without graphics for significant speedup.

Our experimental setup had three main components, the NEAT evolution module, the Simulation adapter and the game itself, which could either be Starcraft II or our own simulation of the game. The evolution module is concerned with running the evolution by assimilating the fitness results and generating networks. We used the SharpNeat implementation of NEAT by Colin Green [33] for the evolution module and adapted it for our purpose. A simulation adapter is the mediator between the evolutionary algorithm and the game which we implemented using sockets to be able to run the game simulations in parallel. It gets the configuration from the NEAT module and sets up the game, it also gets a neural network configuration from the evolution module and feeds inputs with the current game state into the network and uses the output from the network to feed the game and move entities. The adapter returns the final fitness after running the simulation which ends when one of the players has no units left or after a set number of frames. The game component can be either Starcraft or our combat simulation. The architecture diagram of the components is shown in figure 2.

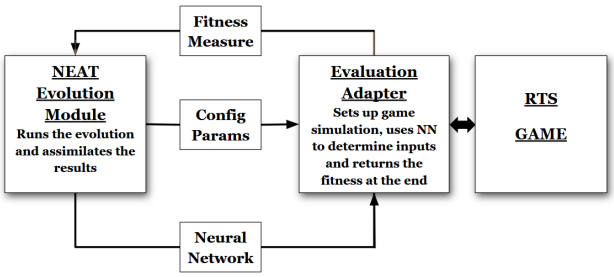


Fig. 2. Architecture Diagram

1) Map configurations: We choose five different unit spawn configurations that determine entity starting positions. These are diagonal, side by side, surround, surrounded and random. In diagonal, opposing sides spawn in groups diagonally on a square map. Similarly, in the side by side configuration, entities appear along the same y-coordinate separated by a fixed distance of 10% of the map size. In surrounded, vultures or hellions appear at the center in a group surrounded by number of zealots and the opposite is true for surround - hellions or vultures surround zealots. We also experimented on random spawning locations for all units of both players. We kept the number of vultures or hellions constant at five (5) but randomly varied the number for zealots for different configurations. In the rest of the paper, we mean vultures or hellions when we use the term ranged units.

2) Fitness Function: We used a fitness function that considers both the damage received and the damage dealt by the our evolving ranged units (vultures and hellions). Over evolutionary time, fitness gradually grows as the ranged units get better at damaging zealots and at evading attacks. At the end of each game run, we sum the remaining hitpoints for both zealots (Hz) and ranged units (Hh) and subtract the remaining hitpoints of zealots from the remaining hitpoints of the ranged units. We add the maximum hitpoints for all zealot units so that the fitness function is always positive.

For number of starting zealots Nz, remaining zealots Rz, remaining hellions Rh, and maximum hitpoint of zealot $Hzmax$, fitness F is calculated as:

$$F = \sum_{n=1}^{Nz} Hzmax_n + \sum_{n=1}^{Rh} Hh_n - \sum_{n=1}^{Rz} Hz_n$$

We should note that as the hit points of both zealots and ranged units are similar, with increasing numbers of zealots and low numbers of ranged units, this fitness function leans towards giving more weight to damage done than damage received. This could be better balanced in various ways. For example, by multiplying the sum of hitpoints of hellions by a balancing factor. Nevertheless, we found that the current configuration performed well during our experimentation phase.

IV. Results and Discussion

We experimented with two different RTS game environments. The Starcraft II game, and our simulation of the RTS environment particularly developed for fast simulation of skirmishes. Below, we describe our experiments and analyze the results for each.

A. Simulation Results

TABLE III
HYPER-PARAMETERS FOR NEAT EVOLUTION

Property	Simulation	Starcraft II
Population	50	50
Generations	100	100
Species	5	5
Initial Conn Probability	0.2	0.1
Elitism Proportion	0.2	0.2
Selection Proportion	0.2	0.2
Asexual Offspring Proportion	0.5	0.8
Sexual Offspring Proportion	0.5	0.2
Inter-species Mating	0.01	0.01
Connection Weight Range	5	7
Probability Weight Mutation	0.95	0.95
Probability Add Node	0.01	0.02
Probability Add Connection	0.025	0.04
Probability Delete Connection	0.025	0.025

In our first set of experiments using our simulation environment, we evolved vultures against a larger group of zealots. Zealots in our simulation, use a hand coded AI which controls each unit as follows: pursue the nearest vulture and attack when it is in range. Both zealots and vultures were given complete map vision - there was no fog of war and thus they did not have to explore the map and could start pursuing their enemy right away. Note that this is a significant difference from SCII.

We ran NEAT on a population size of 50 individuals for 100 generations. The following results are average of 10 different runs of a complete evolutionary epoch, started with different random seeds. Various hyper-parameters that we used for the evolution are noted in table III. Each genome was evaluated based on a complete run of a test configuration, which consisted of 10 different spawning locations with different number of zealots and vultures. We sum the fitness for each of the 10 different training configurations to get the final fitness, which is then forwarded to the NEAT module. We run each scenario until one of the player loses all his units or for a maximum number of frames. We had the option to run our simulation without the graphics rendering which significantly decreased running time compared to SCII.

Initially, we tried to evolve agents only based on a single test configuration of the map, but results showed that they did not generalize well to new scenarios. Using the sum of fitnesses from different configurations led to good generalization across different map configurations and different numbers of units. The 10 different test cases are a sample from the the possible configuration space of different number of zealots and 5 different starting configuration. The training scenarios are listed below and are depicted in Figure 3.

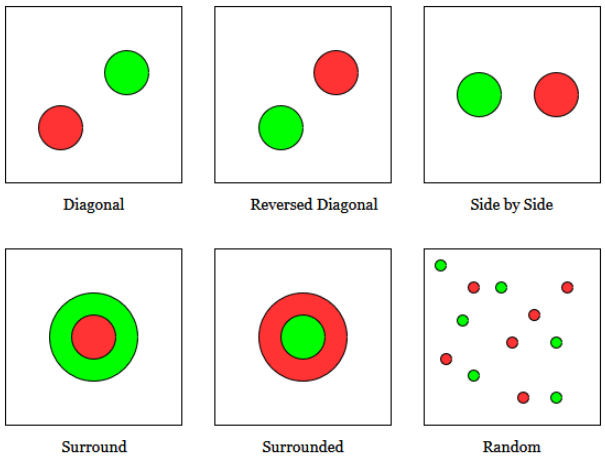


Fig. 3. Map configurations for friendly (green) and enemy (red) positions used for training

1) Diagonal, 25 zealots
2) Reversed Diagonal, 20 zealots
3) Side by Side, 10 zealots
4) Reversed Side by Side, 15 zealots
5) Surround, 20 zealots
6) Surround, 10 zealots
7) Surrounded, 20 zealots
8) Surrounded, 25 zealots
9) Random, 15 zealots
10) Random, 25 zealots

In our simulation environment, the average number of generations over ten runs, needed to find the best individual was 80 and the average best fitness was 96% of the maximum fitness possible. We found that the the evolved vultures learned kiting or to hit and run, against the group of zealots. Kiting is an effective tactic for speedy ranged units against slow melee units as explained earlier.

After evolving neural networks to control vultures with kiting ability against groups of zealots, we tested for the generalizability of our result against scenarios that the Vultures did not encounter during the training phase. For each possible starting configuration, we varied the number of starting zealots from 1 to 30 while the number of Vultures was always constant at 5. Here, we note that Vultures were only evolved against the group of 10, 15, 20 and 25 zealots thus, their performance against different number of zealots shows the robustness of the evolved network.

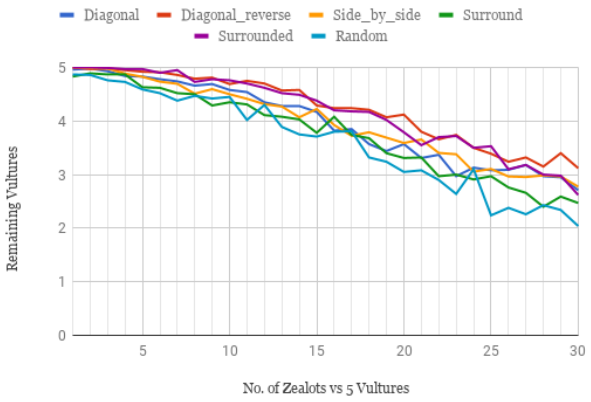


Fig. 4. Remaining Vultures corresponding to increasing zealot numbers

Results of our generalizability tests are shown in Figures 4 and 5. The vertical axis represents the number of units remaining at the end of each game run and the horizontal axis represents the number of zealots against which the five vultures skirmish. We present the results for 6 different starting positions, each starting position averaged over ten runs. Figure 4 shows the number of vultures remaining when times runs out while Figure 5 shows the number of zealots remaining. As shown in Figure 4, we see that vultures' performance smoothly decreases as the number of zealots increases. The number of vultures never goes below two, a good indication of the robustness of evolved networks.

Generalization with respect to damage done is shown in figure 5 where we note that the zealots are completely destroyed by vultures till the number of starting zealots rises above 13. The number of surviving zealots then gradually increases across all our scenarios. The number of remaining zealots never rises above six another good indicator of micro quality and robustness.

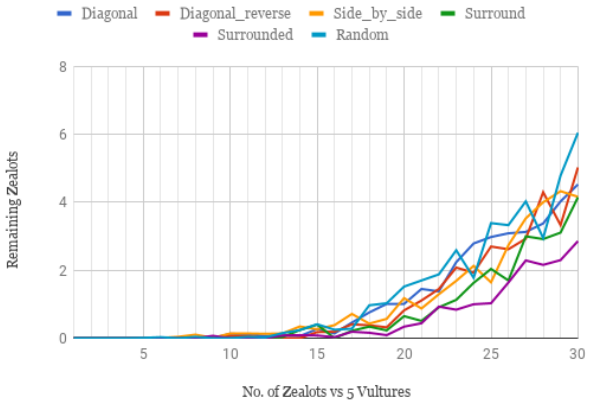


Fig. 5. Remaining zealots corresponding to increasing zealot numbers

We also found that we must provide a number of different training configurations in order to evolve robustness. Evolving with only one configuration, results in much less robust networks whose performance might jump form high to low or low to high when changing the number of zealots even by as little as one zealot. In one case removing a single zealot significantly **decreased** vulture performance. This variability is also shown by other neural network based approaches for Starcraft AI [31].

B. Starcraft II Results

In Starcraft II, we evolved hellions which are a ranged units that can do splash damage against zealots which are strong melee units. We control the movement and attack commands for the hellions after translating the outputs from NEAT evolved neural networks to the commands for Starcraft II using the API. We ran the game at the top speed of 16. As Starcraft II runs relatively slow even at top speed, we ran on 15 machines for 24 hours.

For StarcraftII experiments, we ran on a population size of 50 for 100 generations and the results averaged over 10 times for the final results. We use the same NEAT parameters as for our simulation and given in table III. Unlike our simulation, we used the sum of fitness for only three different configurations to get the individual fitness. The three configurations and corresponding number of zealots are listed below; the number of hellions is always five.

1) Diagonal, 25 zealots
2) Random, 20 zealots
3) Side by Side, 15 zealots

Over 10 runs, the average number of generations needed to find the best individual was 85 and the average best fitness was 88% of the maximum fitness possible. The evolved network also displayed kiting behaviour against the zealots - similar to our findings from the simulation approach.

We tested for the generalizability of the evolved networks in similar fashion to the tests for then simulation environment. That is, we tested the best evolved network against new configurations and with varying number of zealots. Here, we note that hellions only evolved against groups of 15, 20 and 25 zealots, and on only three configurations. Generalization results are shown in Figure 6 and 7. The vertical axis represents the number of units remaining at the end of each game run and the horizontal axis represents the number of zealots remaining when skirmishing with five hellions. We present the results for six different starting positions with the number of zealots varying from 1 to 30. We ran the simulation for five runs on the same starting configuration to get the average number of remaining units. We expect to do more runs as computational resources allow.

As shown in figure 6, we see a downward trend for then number of remaining hellions starting from 5. However, unlike in our simulation, the trends are different for different starting configurations. Diagonal and side by side show good performance across different numbers of zealots while random and surrounded perform comparatively lower. The gradual decrease is expected as the hellions get overwhelmed by the increasing number of zealots. Still, we can see from the graph

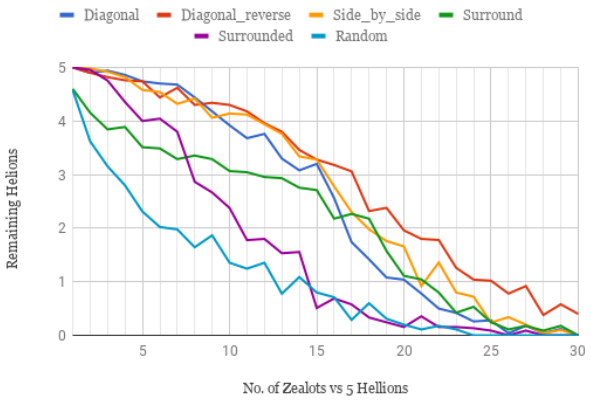


Fig. 6. Remaining hellions corresponding to increasing zealot numbers

that hellions are generalizing well with respect to damage received against different number of zealots and different starting positions.

Generalization with respect to damage done is shown in figure 7 where we again note that the hellions perform well for diagonal and side by side scenarios while performing comparatively lower in random and surrounded scenarios. The number of remaining zealots for each scenario only gradually increases on these previously unseen scenarios and indicates generalizability of our evolved result.

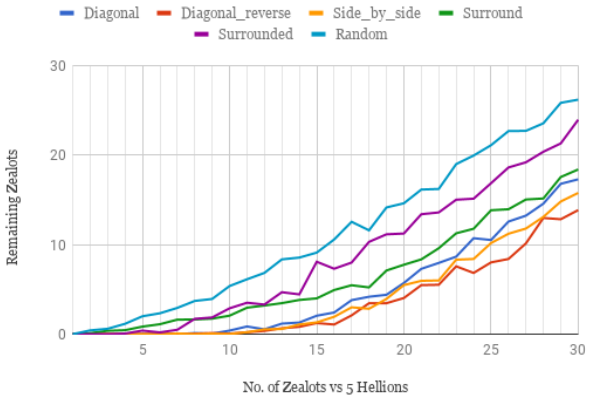


Fig. 7. Remaining zealots corresponding to increasing zealot numbers

C. Comparison of Results

We have shown that the NEAT generated neural networks from from Starcraft II and our simulation were able to generalize with respect to different starting positions and different numbers of zealots. Ranged units performed well against smaller numbers of zealots and performance decreased with increase in number of zealots for both environments. The gradual progression of values for different series without major deviance indicates that the evolved network is robust against changes in both starting position and number of zealots. Videos at https://www.cse.unr.edu/ aavaas/Micro.html serve well to indicate the quality and robustness of the evolved micro.

NEAT was able to evolve kiting behaviour in both our simulation and in the Starcraft II environments but there are some differences between the results from two environments. The evolved network in Starcraft II seemed to perform comparatively worse than our simulation. We believe the fewer training configurations, the increased complexity of SCII, and differences in the AI we evolved against, account for these differences.

V. Conclusion and Future Work

Our research focused on using NEAT to evolve neural networks that could provide robust control of a squad in an RTS game skirmish. We showed that our representation of the game state provided to NEAT sufficed to evolve high performing micro, while training on a variety of scenarios leads to more robust and high performing micro. The evolved networks generalized well to different numbers of opposing units and different starting configurations.

We used our own simulation environment for initial experimentation and exploration. Because our simulation runs much faster than Starcraft II, we were able to explore multiple input and output representations and different evolutionary hyperparameters to hone in good representations and parameters. We then used this experience to try reproduce our results in the popular RTS game. Starcraft II. Here, we ranged hellions evolved kiting behaviour against melee zealots - like in our simulation environment and meeting our expectations. We believe these results show that NEAT holds promise as a potential approach to evolving RTS game micro.

With a general neural network representation and with NEAT, we think that our approach can be effectively extended to approach more complex scenarios and group configurations. We are working on probabilistic activation model for outputs: we can consider the output of the neural network as the probability of it being active rather than comparing it against the threshold. Using recurrent neural networks would enable incorporating sequential strategies spanning multiple time frames. In addition, we will be adding more game state information about opponents, considering a multi-objective formulation of the fitness function, and obtaining and using much larger computational resources.

References

[1] S. Ontanón, G. Synnaeve, A. Uriarte, F. Richoux, D. Churchill, and M. Preuss, "A survey of real-time strategy game ai research and competition in starcraft," *IEEE Transactions on Computational Intelligence and AI in games*, vol. 5, no. 4, pp. 293–311, 2013.

[2] M. Buro, "Call for ai research in rts games," in *Proceedings of the AAAI-04 Workshop on Challenges in Game AI*, 2004, pp. 139–142.

[3] G. Robertson and I. Watson, "A review of real-time strategy game ai," *AI Magazine*, vol. 35, no. 4, pp. 75–104, 2014.

[4] S. Liu, S. J. Louis, and C. A. Ballinger, "Evolving effective microbehaviors in real-time strategy games," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 8, no. 4, pp. 351–362, 2016.

[5] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary computation*, vol. 10, no. 2, pp. 99–127, 2002.

[6] J. H. Holland, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[7] M. Hausknecht, J. Lehman, R. Miikkulainen, and P. Stone, "A neuroevolution approach to general atari game playing," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 6, no. 4, pp. 355–366, 2014.

[8] Blizzard. (2018). Starcraft2 client library designed for building scripted bots and research, [Online]. Available: https://github.com/Blizzard/s2client-api.

[9] M. Buckland and M. Collins, *AI techniques for game programming*. Premier press, 2002.

[10] S. Rabin, *AI Game programming wisdom 4*. Nelson Education, 2014, vol. 4.

[11] B. G. Weber and M. Mateas, "A data mining approach to strategy prediction," in *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*, IEEE, 2009, pp. 140–147.

[12] D. Churchill, A. Saffidine, and M. Buro, "Fast heuristic search for rts game combat scenarios.," in *AIIDE*, 2012, pp. 112–117.

[13] S. Wender and I. Watson, "Applying reinforcement learning to small scale combat in the real-time strategy game starcraft: Broodwar," in *Computational Intelligence and Games (CIG), 2012 IEEE Conference on*, IEEE, 2012, pp. 402–408.

[14] A. Shantia, E. Begue, and M. Wiering, "Connectionist reinforcement learning for intelligent unit micro management in starcraft," in *Neural Networks (IJCNN), The 2011 International Joint Conference on*, IEEE, 2011, pp. 1794–1801.

[15] O. Vinyals, T. Ewalds, S. Bartunov, P. Georgiev, A. S. Vezhnevets, M. Yeo, A. Makhzani, H. Küttler, J. Agapiou, J. Schrittwieser, *et al.*, "Starcraft ii: A new challenge for reinforcement learning," *arXiv preprint arXiv:1708.04782*, 2017.

[16] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *The international journal of robotics research*, vol. 5, no. 1, pp. 90–98, 1986.

[17] J. Hagelback and S. J. Johansson, "Using multi-agent potential fields in real-time strategy games," in *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 2*, International Foundation for Autonomous Agents and Multiagent Systems, 2008, pp. 631–638.

[18] M. Buro, "Orts - a free software rts game engine," *Accessed March*, vol. 20, p. 2007, 2007.

[19] A. Uriarte and S. Ontanón, "Kiting in rts games using influence maps," in *Eighth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2012.

[20] P. Avery and S. Louis, "Coevolving influence maps for spatial team tactics in a rts game," in *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, ACM, 2010, pp. 783–790.

[21] C. Miles and S. J. Louis, "Co-evolving real-time strategy game playing influence map trees with genetic algorithms," in *Proceedings of the International Congress on Evolutionary Computation, Portland, Oregon*, IEEE Press, 2006.

[22] B. Allen and P. Faloutsos, "Complex networks of simple neurons for bipedal locomotion," in *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, IEEE, 2009, pp. 4457–4462.

[23] L. Cardamone, D. Loiacono, and P. L. Lanzi, "Evolving competitive car controllers for racing games with neuroevolution," in *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, ACM, 2009, pp. 1179–1186.

[24] F. G. Durán, F. L. Largo, M. P. López, and R. R. Aldeguer, "Driving bots with a neuroevolved brain: Screaming racers," *INTELIGENCIA ARTIFICIAL*, 2005.

[25] J. Schrum and R. Miikkulainen, "Evolving multimodal behavior with modular neural networks in ms. pacman," in *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, ACM, 2014, pp. 325–332.

[26] L. E. Gillespie, G. R. Gonzalez, and J. Schrum, "Comparing direct and indirect encodings using both raw and hand-designed features in tetris," 2017.

[27] T. Boris and Š. Goran, "Evolving neural network to play game 2048," in *Telecommunications Forum (TELFOR), 2016 24th*, IEEE, 2016, pp. 1–3.

[28] K. O. Stanley and R. Miikkulainen, "Evolving a roving eye for go," in *Genetic and Evolutionary Computation Conference*, Springer, 2004, pp. 1226–1238.

[29] J. K. Olesen, G. N. Yannakakis, and J. Hallam, "Real-time challenge balance in an rts game using rtneat," in *Computational Intelligence and Games, 2008. CIG'08. IEEE Symposium On*, IEEE, 2008, pp. 87–94.

[30] I. Gabriel, V. Negru, and D. Zaharie, "Neuroevolution based multi-agent system with ontology based template creation for micromanagement in real-time strategy games," *Information Technology and Control*, vol. 43, no. 1, pp. 98–109, 2014.

[31] J. S. Zhen and I. D. Watson, "Neuroevolution for micromanagement in the real-time strategy game starcraft: Brood war.," in *Australasian Conference on Artificial Intelligence*, Springer, 2013, pp. 259–270.

[32] (2018). Starcraft vulture, [Online]. Available: http://liquipedia.net/starcraft/Vulture.

[33] C. Green. (2018). Sharpneat, [Online]. Available: http://sharpneat.sourceforge.net/.

Tabular Reinforcement Learning in Real-Time Strategy Games via Options

Anderson R. Tavares and Luiz Chaimowicz
Laboratory of Multidisciplinary Research in Games
Universidade Federal de Minas Gerais
Belo Horizonte, Brazil
{anderson,chaimo}@dcc.ufmg.br

***Abstract*—Real-Time Strategy (RTS) games are complex domains with huge state and action spaces. In such games, humans usually pursue long-term plans, which take long sequences of actions to achieve. In this work, we implement this behavior by reasoning over options, which are temporally-extended actions in Markov Decision Processes. Our options are defined with the aid of a state aggregation scheme and a portfolio of game-playing algorithms. Experimentally, we show that our approach leverages the capabilities of traditional reinforcement-learning techniques, which become competitive against state-of-the-art search methods in μRTS.**

***Index Terms*—real-time strategy games, options, reinforcement learning**

I. Introduction

To succeed in a real-time strategy (RTS) game, a player must handle resource collection, construction of buildings and battles against enemies. Such scenarios are challenging for Artificial Intelligence because of their dynamic nature, their huge state-action spaces and the need for both short- and long-term decision-making.

Part of the research effort in real-time strategy games focuses on search approaches that either operate in abstract representations of a game, as in [1], [2] or are tested in combat scenarios, as in [3]–[5]. Some approaches deal with RTS games as a whole, as in [6]–[8]. However, search approaches require a forward model of the world. Supervised learning approaches, as [9], dismiss the forward model, but require labeled data. Reinforcement learning [10] does not need labeled data, but so far it has been restricted to combats, as in [11]–[13], which are mainly reflex-based, not requiring long-term decisions. In other words, reinforcement learning approaches are unable to handle all aspects of real-time strategy games successfully, due to their complexity.

This paper contributes with a methodology that allows the application of classic reinforcement learning approaches to problems with large state and action spaces, including real-time strategy games, in all its aspects. We require (1) a state abstraction function and (2) a portfolio of domain-specific algorithms, scripts or heuristics, that receive the current environment state and output a valid action. Those are combined to allow the use of *options* [14], or temporally-extended actions in Markov Decision Processes. We create an option for every abstract state and algorithm in the portfolio. Our learning agent observes the abstract state and selects an option, which acts according to its associated algorithm. The option terminates upon reaching a new abstract state. The learning agent then observes the new state, the received reward, and selects a new option, just as in the traditional tabular reinforcement learning framework, but now applied to an abstract representation.

We instantiate the proposed approach in μRTS [15], using simple game-playing algorithms to define the options' policies. Our approach enables classic Q-learning [16] to play the game as a whole, not only combats. It achieves competitive performance against some state-of-the-art search approaches by reasoning at a different temporal scale via the options framework. We also identify limitations regarding the proposed state abstraction, which is still challenging for the agent, and the reward function, that directs the agent to a myopic behavior, sacrificing long-term benefits in favor of short-term material advantage.

This paper is organized as follows: Section II discusses related work; Section III introduces our framework; Section IV describes μRTS and the instantiation of our approach; Section V shows our experiments against state-of-the-art search approaches; Section VI presents an overall discussion with the framework benefits, its limitations and potential solutions; and Section VII presents concluding remarks and directions for future research.

II. Related work

Our use of options in real-time strategy games is associated with game-playing algorithms, also called strategies or scripts. Many works in the literature use game-playing algorithms to abstract from the game's low-level actions. For example, [17] use fuzzy rules to determine usefulness of strategies according to the game state in StarCraft; [18] use case-based reasoning in Wargus, an open-source Warcraft II clone; [19], [20] study game-theoretic approaches for strategy selection and even StarCraft bots rely on a pool of build-orders, or scripts to dictate early-game behavior, to play a match [21]. However, those approaches do not make use of reinforcement learning frameworks as a mean to learn useful behaviors, as in the

Authors are thankful to CAPES, CNPq and FAPEMIG for support in this research.

978-1-5386-4359-4/18/$31.00 ©2018 IEEE

present work. Reinforcement learning has the advantage of not requiring expert knowledge to determine the usefulness of a given behavior, as the agent learns from experience, in contrast with the fuzzy approach of [17], nor it requires forward models of the world to perform planning, as in [20].

Our use of options is equivalent to selecting a single algorithm that controls all units in a real-time strategy game. A similar, but finer-grained approach is to assign scripts individually to game units: [3] perform the assignment via hill-climbing, which [4] later accelerates with a type system and [5] use an online evolutionary algorithm for the assignment. Although assigning individual unit scripts has potential for achieving strong game-playing behavior, it has only been tested in combat scenarios of real-time strategy games, and with approaches that require the game's forward model to simulate the assignments' outcome.

Model-free reinforcement learning approaches have been successfully applied to combat scenarios in real-time strategy games. For example, [11] use a tabular approach, partitioning the game state space into clusters of states with similar features - an idea we use for our abstract state representation (see Section III). The tabular approach of [11] is able to tackle small-scale combat. Other works, such as [12] and [13] use deep reinforcement learning approaches, in which a raw state representation is fed to a convolutional neural network. From a large number of samples, such networks are able to extract state features, generalizing action-value estimates. Nevertheless, [12] and [13] tackle only the combat task in real-time strategy games, albeit in a larger scale than [11]. Combats are reflex-based tasks, where the player must promptly react to the current elements on screen. Real-time strategy games as a whole encompass long-term planning in tactical and strategic levels as well, which are not tackled in combat-centered approaches.

In parallel to real-time strategy games, deep reinforcement learning has tackled classic Atari games in the Arcade Learning Environment [22], with similar successes and struggles as in real-time strategy games research. The approach of [23] achieves super-human performance in reflex-based games, but performs poorly in games with long-term planning horizons, such as *Montezuma's Revenge*. A two-level hierarchical approach is introduced in [24] to enable reasoning at different time scales. The upper layer observes the environment state and chooses a goal, and the lower layer learns a policy to achieve that goal (i.e., the option's policy). The upper layer receives the environment reward and generates intrinsic reward signals to the lower layer. The proposed approach remedies the deficiency of [23] in Montezuma's Revenge, using manually-designed goals. Posteriorly, [25] propose an approach to discover options based on intrinsic reward functions that direct the agent towards traversing the state space in directions specified by a learned representation. The proposed approach was able to discover similar options to the ones achieved via the handcrafted goals of [24].

Real-time strategy is much more complex than Atari: for a given state, there are 18 actions in Atari (all possible button combinations), whereas a single real-time strategy game unit can perform a similar number of actions. Each player controls several units, resulting in a combinatorial explosion of possible actions. Nevertheless, the use of options has remedied the deficiency of deep reinforcement learning in Atari games, by allowing reasoning at different time scales. Thus, we bring this idea to real-time strategy games, albeit in a tabular reinforcement learning framework.

III. Options in Real-Time Strategy Games

Real-time strategy (RTS) games are sequential decision problems in large state and action spaces. Formally, we can model a RTS game as a discrete Markov Decision Process[1] (MDP), which is defined by a tuple (S, A, T, R):

- S is the set of environment states;
- A is the set of actions;
- T: $S \times A \times S \rightarrow [0, 1]$ is the state transition function. It indicates the probability of reaching a state, given an action taken in a previous state;
- R: $S \times A \rightarrow \mathbb{R}$ is the expected reward function. It indicates the expected reward of taking an action in a given state.

The agent interacts with a MDP according to its policy $\pi : S \times A \rightarrow [0, 1]$, which indicates the probability of taking an action in a given state. An usual goal in a MDP is to maximize the expected sum of discounted rewards: $E\left[\sum_{j=0}^{\infty} \gamma^j r_{t+j}\right]$, where t is the current time, r_{t+j} is the reward received j steps in the future and $\gamma \in [0, 1]$ is a discount factor indicating how much the agent values future rewards.

Tabular reinforcement learning methods, such as Q-learning [16], maintain estimates of action-values in tabular form: each entry $Q(s, a)$ indicates the expected discounted sum of rewards of taking action a in s and following the optimal policy thereafter. Given an experience tuple $\langle s, a, r, s' \rangle$, which indicates that the agent took action a in state s, receiving reward r and reaching state s', Q-learning updates the action-value estimates via Eq. 1, where α is the step-size or learning rate, which indicates by how much the current action-value estimate moves towards the sampled value.

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_{a' \in A} Q(s', a') - Q(s, a) \right] \quad (1)$$

In real-time strategy games, the state and action spaces are huge. For example, in StarCraft, [21] estimated 10^{1685} states and from 10^{50} to 10^{200} actions. Such complex domains are impractical for tabular reinforcement learning approaches.

To tackle the complexity of a real-time strategy game, we resort to *options* [14], which are temporally-extended actions in Markov Decision Processes. Intuitively, an option is a sequence of actions, which might aid the learning agent to achieve a specific goal. Formally, an option is defined by a tuple $\langle I, \pi, \beta \rangle$. $I \subseteq S$ is the initiation set: an option is available

[1]Real-time strategy games are adversarial domains, but by modeling the domain as an MDP we are interpreting the opponent as part of the environment. Explicitly accounting for the adversary would require a model with joint actions [26], which is out of this papers' scope.

in state s if and only if $s \in I$. The option's policy is π: if the option is active in state s, actions are selected according to the probability distribution dictated by $\pi(s, \cdot)$. The termination condition $\beta : S \rightarrow [0, 1]$ indicates the probability of an option terminating in a given state.

Many real-time strategy games have scripts or algorithms, either built-in or developed by enthusiasts. The algorithms define game-playing policies, mapping game states to actions. Thus, we denote an algorithm by the policy π it specifies. We then need to define the remaining components of the options: their initiation sets and termination conditions. In this work we do so with via a state abstraction scheme, where we partition the set S of primitive states into a set $\overline{S}$ of clusters, or abstract states. A state abstraction function $\phi : S \rightarrow \overline{S}$ maps a state in the primitive state space to an abstract state[2].

Given the abstract state set $\overline{S}$, the corresponding state abstraction function ϕ and our portfolio of algorithms Π, the set of options is defined formally as follows: for each abstract state $\overline{s}$ and algorithm $\pi \in \Pi$, there is an option $o = \langle I, \pi, \beta \rangle$ with initiation set defined as: $I = \{s \in S : \phi(s) = \overline{s}\}$ and termination condition defined as: $\beta(s) = 0$ if $s \in I$ and $\beta(s) = 1$ otherwise. In other words, for each abstract state $\overline{s}$, there are $|\Pi|$ available options. The agent then selects an option o, which performs actions according to the corresponding policy, passing through various primitive states, and terminates when the current primitive state maps to a different abstract state from the one it has initiated at.

A reinforcement learning agent reasoning over our options scheme can use an option-value function $Q(\overline{s}, o)$, which indicates the expected sum of discounted rewards of taking option o at the abstract state $\overline{s}$ and then following the optimal policy over options thereafter. The option selection agent, upon reaching an abstract state $\overline{s'}$ and receiving a reward r after selecting option o at state $\overline{s}$, can update its option-value estimate via Eq. 2, which is the Q-learning update rule of Eq. 1 adapted to options.

$$Q(\overline{s}, o) \leftarrow Q(\overline{s}, o) + \alpha \left[r + \gamma \max_{o' \in O_{s'}} Q(\overline{s'}, o') - Q(\overline{s}, o) \right] \quad (2)$$

Our approach for learning over options is formalized in Alg. 1, where $O_{\overline{s}}$ is the set of options available at the abstract state $\overline{s}$. The ϵ-greedy selection means to select a random option with probability ϵ and the greedy, value-maximizing option ($\text{argmax}_{o \in O_{\overline{s}}} Q(\overline{s}, o)$) with probability $1 - \epsilon$.

Our agent uses only the reward of the transition that led it to a different abstract state to update the option-value function Q, whereas in the original definition [14], the agent accumulates all rewards received in the trajectory determined by the selected option. This modification suits the idea that the information received when the agent reaches a different abstract state is more important, because primitive states in the same cluster are similar. Nevertheless, the algorithm can be easily changed to accommodate the original definition, by accumulating the rewards received during the option execution (lines 5–9).

Algorithm 1 Learning over Options

```
1:  s ← initial state
2:  s̄ ← φ(s)
3:  while s̄ is not terminal do
4:      Choose option o = ⟨I, π, β⟩ ∈ O_s̄ via ε-greedy
5:      while s ∈ I do
6:          Select action a according to π(s, ·)
7:          Execute a in s, observe reward r and next state s'
8:          s ← s'
9:      end while
10:     s̄' ← φ(s')
11:     Use ⟨s̄, o, r, s̄'⟩ to update Q via Eq. 2.
12:     s̄ ← s̄'
13: end while
```

Our agent executes an option until termination (lines 5–9 in Alg. 1) and our set of options do not include primitive actions[3]. Formally, according to Theorem 1 of [14], the agent follows a Semi-Markov Decision Process (SMDP), which adds important implications: for example, an optimal policy over options does not necessarily result in optimal behavior in the original MDP. Intuitively, this happens because the agent is limited by the options' policies, as they perform the primitive actions. In complex domains such as real-time strategy games, sacrificing optimality in exchange for feasibility is a reasonable choice.

The success of learning over options in real-time strategy games thus depends on the quality of the available options' policies, which depend on the portfolio of algorithms and on the quality of the proposed state abstraction scheme. Coarser abstractions may ignore important context information. An extreme case is [19] which considers a single abstract state or decision point, ignoring all context information. On the other hand, finer abstractions result in more abstract states, which increase the number of entries in the option-value function, requiring more training to learn useful policies.

IV. Scenario

A. Testbed

We evaluate our learning over options approach in μRTS (Fig. 1), a real-time strategy (RTS) game designed to facilitate artificial intelligence research [15]. It simplifies RTS games by having fewer types of units and buildings, a shallow technology tree, and simpler combat model. Moreover, μRTS provides a forward model to foster the development of search and planning methods, some of which are our adversaries in the experiments.

In μRTS, there are two types of buildings: *Bases* and *Barracks*, which produce *Workers* and military units, respectively. Workers harvest *resources* (needed to create buildings and produce units), construct buildings and have limited melee

[2]We follow the notation of [27] to represent the set of abstract states and the abstraction function.

[3]Primitive actions can be seen as one-step options [14] and thus could be included in the set of options as well.

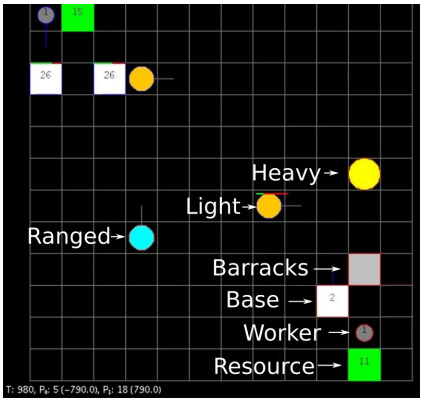


Fig. 1. A screenshot of μRTS.

combat ability. Military units are either *Heavy*, *Light* or *Ranged*. Heavy and Light are strong but slow and weak but fast melee units, respectively. Ranged are weak units, but can attack from distance. To win a μRTS match, a player must destroy all buildings and kill all adversary units. In this work, μRTS is configured with full visibility (perfect information) for the players.

B. Model instantiation

This section discusses our model of options in μRTS as well as the reward function we adopt to train our learning agent.

When μRTS is configured for full visibility (perfect information), a software-controlled player has access to all game data: both players' resources, attributes and actions of all units and buildings, and even information on units and buildings under production, including the remaining time to produce. Thus, the μRTS game state has the Markov property, as there is no underlying information hidden from the player.

Our options are defined with abstract states and a set of algorithms (see Section III). Our set $\overline{S}$ of abstract states is constructed by identifying eight features in μRTS. First, we divide the game duration equally into five intervals, related to the match stage: opening, early, mid, late and end. The other seven features account for material advantage between player and opponent in each game entity: resources, bases, barracks, workers, heavy, light and ranged units. As a simplification, the material advantage features are discretized in three values: *ahead*, *even* and *behind*, assigned as follows: ahead if the player controls two or more entities of the given type than the adversary; even if the difference is between +1 and -1 and behind if the adversary controls two or more entities than the player. In total, we have $5 \times 3^7 = 10935$ non-terminal abstract states. A terminal is created as an additional abstract state. It is reached whenever a player wins the game or time runs out. That is, all primitive terminal states in μRTS are mapped to the abstract terminal state. Our state abstraction function ϕ then maps a primitive μRTS state to an abstract state by checking the game termination conditions, and then determining the stage from the current game time and counting the entities of each type owned by the player and its opponent to determine the features' values.

Our abstract state representation does not have the Markov property: from a given abstract state, different abstract states could be reached under the same option, depending on which primitive state the player is actually in. However, our learning agent overlooks this issue, acting as if the states were markovian.

Our portfolio Π of algorithms contains four simple built-in μRTS scripts: *Worker*, *Ranged*, *Heavy* and *Light* rushes plus two scripts we developed: *Expand* and *BuildBarracks*. Worker rush only trains workers, using one to harvest resources and the remaining to attack the adversary. The other rushes use a worker to construct a single barracks which trains units of the respective type to attack the adversary. The single worker keeps harvesting resources after constructing the barracks. Expand and BuildBarracks do not perform combat actions themselves. Instead, they send a worker to build a new base or barracks, allowing faster production of workers or military units, respectively.

Our reward signal is the player score minus the opponent's. The score function counts the material a player has. The function is provided by μRTS, being used as a state evaluation function by some search approaches. The score of a player i is calculated according to Eq. 3, where B_i is the amount of resources owned by the player, W_i is the amount of resources being carried by player i's workers, U_i are the units player i controls, $c(u)$ is the cost to produce a unit, $hp(u)$ are the current hit points of a unit and $hp_{max}(u)$ are the maximum hit points of a unit.

$$\text{score}_i = 20 \cdot B_i + 10 \cdot W_i + \sum_{u \in U_i} 40 \cdot c(u) \cdot \sqrt{\frac{hp(u)}{hp_{max}(u)}} \quad (3)$$

V. Experiments

In our experiments, we run μRTS matches with 3000 frames of duration on the map "basesWorkers24x24". Each run of our approach consists of 5000 training episodes against an adversary, where each episode is a μRTS match. We then extract the resulting policies and run 100 test matches against the same adversary. Parameters were determined from preliminary experiments as follows: α decays exponentially with rate 0.99907939, (from 1 to 0.01 in 5000 episodes); $\gamma = 0.9$ and $\epsilon = 0.1$. The search methods (our adversaries) were executed with their default parameters in μRTS. We run 5 repetitions of each experiment. All experiments were executed in a 40-core machine with 256GB of RAM memory. Each core is a Intel(R) Xeon(R) CPU E5-2650 v3 @ 2.30GHz.

We use the following state-of-the-art search methods as our opponents to evaluate our learning over options approach:

- Strategy simulation, or SS for short [20]: at each frame, playouts among scripts are used to fill a normal form game's payoff matrix. Then, Nash Equilibrium is calculated and a script is selected accordingly. The available scripts are the same we use, except by BuildBarracks and Expand, that have no combat capabilities.

- Adversarial Hierarchical Task Network, or AHTN for short [7]: this approach extends the use of hierarchical-task network (HTN) planning, which encode domain knowledge via the definition of useful tasks in the domain, with a minimax-like game-tree search.
- Puppet-$\alpha\beta$ [6]: PuppetSearch is a framework that augments the capabilities of game-playing scripts by means of move choices they expose to search procedures. By exposing a restricted set of actions, the search methods can look further ahead, finding potentially better solutions. Moreover, as computational budget allows, more choices can be exposed so the search algorithms can investigate with a broader perspective. In [6], StarCraft playing scripts are combined with a version of the α-β considering durations (ABCD) algorithm [28]. This approach was ported to μRTS as well.
- PuppetUCT [8]: this is a variant of Puppet-$\alpha\beta$, which replaces the ABCD algorithm with a version of the upper-confidence bound for trees considering durations (UCTCD) algorithm [3].

Figure 2 shows the training performance of Q-learning over options against each adversary, in terms of mean reward (a) and win rate (b) on the 5 repetitions. We show the running average on 100 episodes do display the overall trend of the training process. The baseline for the reward is zero: a superior value means that the player had more material than its opponent throughout the match. The baseline for the win rate is 0.5: a superior value means that the learning agent has defeated its adversary in most repetitions.

Against all opponents, rewards (Fig. 2a) become positive after a number of episodes. As the reward function reflects material advantage, the agent is being able to maintain more units than its opponents during parts of the game. However, the plots show the rewards accumulated for an entire episode. It might be the case that the agent maintains material advantage for a period in the game but loses the match near its end. In this case, the reward accumulated in that episode might still be positive. Moreover, the rewards oscillate through the entire training period, even with the running averages. This might be due to actions of the adversary: the agent might learn a policy that reaches states where the search opponent performs well, reducing the agent reward, forcing it to change its policy again. It is also possible that the agent could learn more stable and stronger policies by training for more episodes. Section VI discusses the issue of the number of training episodes in more detail.

The win rate during training (Fig. 2b) increases against all opponents, although the increment and final performances are weaker against PuppetUCT and Puppet-$\alpha\beta$ than against AHTN and SS. The learning agent was able to win the majority of matches against AHTN and SS, getting above the baseline, but has not succeeded against PuppetUCT and Puppet-$\alpha\beta$, remaining below the baseline during the entire training. This suggests that our approach was able to learn strong policies against AHTN and SS, but was unable to do the same against PuppetUCT and Puppet-$\alpha\beta$. Moreover, this indicates that the reward function is not directing the agent towards victories. The agent is likely acting myopically: even if it loses a match in the end, receiving a few negative rewards, it might be taking decisions that maximize its immediate reward in earlier game stages, ending up with a positive reward balance.

To evaluate resulting policies' strength, we run 100 test matches against the same training opponents. In the tests, ϵ and α are set to zero, to ensure the agent acts greedily according to the learned policy, without further updates. As a baseline, we also tested random and fixed policies over options against each adversary. Each fixed policy always chooses a single option. We tested a fixed policy for each available option with combat capabilities: Light, Heavy, Worker and Ranged rushes. Figure 3 shows the results[4]. Heavy rush is omitted because it does not win any match in the tests.

The resulting policy of Q-learning outperforms a random policy over options against all adversaries. Interestingly, however, for each adversary, there is a fixed policy over options that outperforms the Q-learning resulting policy. This means that each adversary can be defeated by a simple script. Nevertheless, learning is useful, because no fixed policy defeated all search approaches. Thus, an agent must learn which option (or combination of options) defeats each adversary.

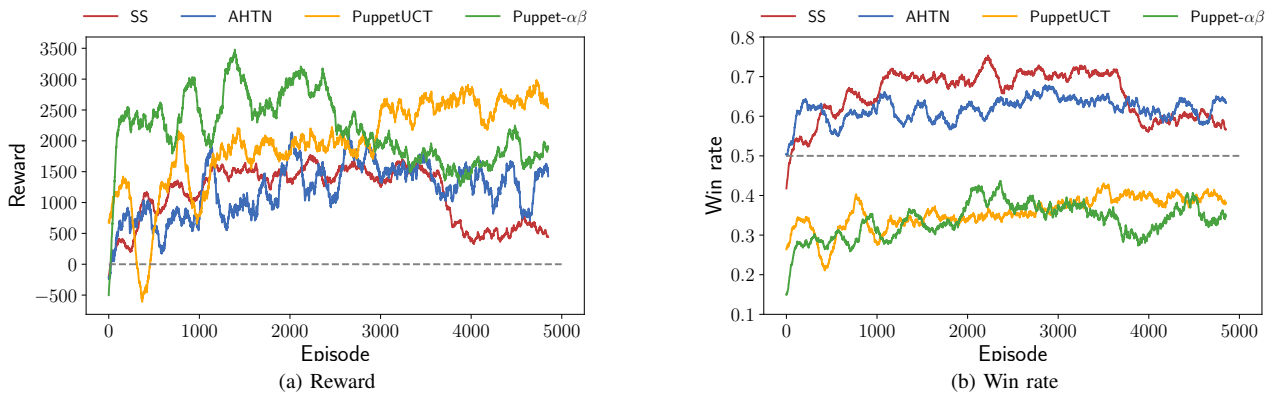
Our approach learned a strong policy against SS, competitive policies against AHTN and PuppetUCT, and a poor policy against Puppet-$\alpha\beta$. Chronologically, SS [20] was proposed earlier, followed by AHTN [7] and then by Puppet-$\alpha\beta$ [6] and PuppetUCT [8]. Our approach fared better against older approaches rather than newer ones, except by Puppet-$\alpha\beta$ which is older than PuppetUCT and imposed more difficulties to our approach.

VI. Discussion

Our approach enabled the use of Q-learning, a traditional tabular reinforcement learning method, in a real-time strategy game. By reasoning over options in an abstract state space, the learning agent does not need to consider all possible underlying game actions, nor does it need to store action-values for every primitive state. Our approach thus sacrifices optimality to handle a previously unfeasible domain. Our results showed that a random policy over options had poor performance and no fixed policy over options was able to overcome all opponents. Being able to learn from experience has proved essential to success against a variety of adversaries in such a complex domain.

Our model has the limitation of ignoring adversary actions, by interpreting the opponent as part of the environment. Thus, the agent learns a policy suited against the specific training opponent. In theory, such policies could be arbitrarily exploited by an agent employing game-theoretic reasoning. Thus, explicitly accounting for opponent actions [26] or incorporating opponent models [30], [31] would facilitate the discovery of

[4]The search approaches are sensitive to the underlying hardware. The performance of the individual fixed policies reported here against the search approaches has slight differences, for example, with [29].



(a) Reward (b) Win rate

Fig. 2. Running average on 100 episodes of mean reward and win rate in the 5 repetitions against each training adversary.

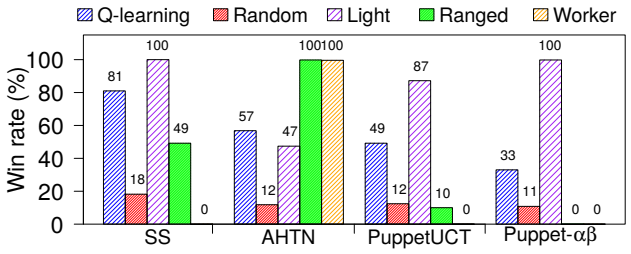


Fig. 3. Victory rate of Q-Learning resulting policy, plus random and fixed policies (Light, Ranged, Worker) over options vs each search method. A policy with missing bar has zero victories against the specific adversary.

more general and/or robust policies. With opponent models, the agent would be able to identify the opponent given its behavior and then learn a proper response.

Our approach demonstrated limited performance against some of the state-of-the-art search methods, even though the agent tried to learn a specialized policy. On the other hand, our reinforcement learning approach dismisses forward models and decides much faster than search approaches, because its decision is an ϵ-greedy procedure, followed by the chosen algorithm's action selection. An ϵ-greedy choice over primitive actions would be impacted by the number of such actions, but ours operate over options, whose quantity is small. The choice of simple algorithms to compose the portfolio was of benefit, because more sophisticated but slower algorithms would result in increased training time.

The first point for improving the performance of our approach would be the adoption of a new reward function, because the current one, related to material advantage, seemed to direct the agent to a myopic behavior. The final game result is the ultimate measure of performance, but it is sparse: the agent must go through all game states to finally receive a feedback of its performance. Thus, learning which choices contributed to the final result is more difficult. This requires more training episodes until the result is gradually fed back to states in earlier game stages. However, training against search methods consumes considerable time because the game speed is limited by the time those approaches need to calculate their actions[5].

Three measures could be adopted to accelerate the agent's training: the use of a simpler state abstraction model, a learning generalization technique and training via self-play.

The current state aggregation model, although enabling the adoption of tabular Q-learning, still has a large number of abstract states (10935 as discussed in Section IV-B) compared to the number of training episodes (5000, in Section V). Although many states will never be reached in real games, the number of possible states is large so that some state-action pairs will have only a few visits to produce precise estimates. A simpler model, on the other hand, could ignore important information, preventing the agent to discover strong policies.

Thus, a learning generalization technique would be of great value: the learned value of a state would be propagated to similar ones. Such idea has achieved remarkable success in Backgammon [32] and Go [33]. A generalization technique specially suited for tabular learning methods is presented in [34]. This could be readily applied, preserving our tabular state abstraction scheme.

Training the agent against a copy of itself (self-play) would be fast, but it does not have mathematical guarantees of convergence, as our agent does not account for the adversary [26]. A model that explicitly accounts for the opponent, as proposed in [26], has those convergence guarantees. Thus, training the opponent-aware agent in self-play has the potential to discover strong game-playing policies, although the complete model scales poorly: the action-value function becomes a joint-action-value function. Thus, the agent must learn the value of his actions for each opponent's action. The number of entries in a tabular joint-action-value function thus increases exponentially with the number of actions of both the agent and the opponent.

[5]Training against AHTN takes about one day, whereas against PuppetUCT it takes about a week.

VII. Conclusion

A. Overview

In this paper we modeled a real-time strategy game as a Markov Decision Process (MDP) and presented an approach based on options to tackle the complexity of the resulting MDP. Options can be seen as specific behaviors that perform sequences of actions in the MDP, aiding the learning agent to navigate through the MDP's state space. We partition the MDP's state space into clusters, or abstract states, which are observed by the learning agent. Then, instead of low-level actions, the agent selects options. Those options determine trajectories in the MDP and terminate upon reaching a different abstract state.

The proposed framework reduces the problem complexity, enabling the use of traditional tabular reinforcement learning methods in real-time strategy games. Our experiments in μRTS demonstrate this. We use simple scripts to define the options' behaviors and our learning agent is competitive against some of the state-of-the-art search methods, which have the advantage of a forward model for planning their actions. Although the simple scripts define strong game-playing policies, they alone are not sufficient to defeat the range of tested opponents, which shows the usefulness of learning and adapting.

Our experiments also revealed limitations on our approach, as it does not learn strong playing policies against some adversaries. We identified that the abstract state representation is still challenging, as the number of states is high compared to the number of training episodes. Training for more episodes would take considerable time as the game speed is limited by the opponents' search methods. The search methods work in real-time, but are considerably slower than the quick look-up performed by the learning agent's decision. We also noted that the reward function is directing the learning agent towards a myopic behavior: it sacrifices long-term benefits in favor of short-term material advantage.

B. Future work

Future work can address the limitations of our approach. Adopting a more realistic reward function (e.g. the final game result) would ultimately direct the agent to win matches, although the outcome of an action taken early in the game is not immediately fed back to the agent. Thus, simply changing the reward function could require more training episodes, which are time-consuming because the training opponents take time to perform their searches.

Learning generalization techniques could accelerate training, by propagating the sampled value of a state to similar states, even in tabular representations [34]. Training the agent against a copy of itself (self-play) would be fast, but our model does not have mathematical guarantees of convergence in such case. Those guarantees are attained by explicitly accounting for the adversary [26]. Alternatively, we could employ function approximation techniques, feeding a neural network with the state representation so that it automatically generalizes learned values to similar states. This idea has been successful in earlier [32] and recent times [33], with the added benefit that we do not need to manually identify state features as we did here.

Future research could also investigate the discovery of options in real-time strategy games, extending the tests of [25], which were performed in Atari games.

Acknowledgment

Authors would like to thank the anonymous reviewers for their thoughtful observations that greatly improved this papers' quality. We also thank Leandro Soriano Marcolino and Sivasubramanian Anbalagan for fruitful discussions on this paper's concepts and results.

References

[1] M. Stanescu, N. A. Barriga, and M. Buro, "Hierarchical Adversarial Search Applied to Real-Time Strategy Games." in *AAAI Conference on Artificial Intelligence in Interactive Digital Entertainment (AIIDE)*, 2014, pp. 66–72.

[2] A. Uriarte and S. Ontañón, "Improving Monte Carlo Tree Search Policies in StarCraft via Probabilistic Models Learned from Replay Data," in *AAAI Conference on Artificial Intelligence in Interactive Digital Entertainment (AIIDE)*, 2016, pp. 100–106.

[3] D. Churchill and M. Buro, "Portfolio Greedy Search and Simulation for Large-Scale Combat in StarCraft," in *IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2013, pp. 1–8.

[4] L. H. Lelis, "Stratified Strategy Selection for Unit Control in Real-Time Strategy Games," in *International Joint Conference on Artificial Intelligence (IJCAI)*, 2017, pp. 3735–3741.

[5] C. Wang, P. Chen, Y. Li, C. Holmgård, and J. Togelius, "Portfolio Online Evolution in StarCraft," in *AAAI Conference on Artificial Intelligence in Interactive Digital Entertainment (AIIDE)*, 2016, pp. 114–120.

[6] N. A. Barriga, M. Stanescu, and M. Buro, "Puppet Search: Enhancing Scripted Behavior by Look-Ahead Search with Applications to Real-Time Strategy Games," in *AAAI Conference on Artificial Intelligence in Interactive Digital Entertainment (AIIDE)*, 2015.

[7] S. Ontañón and M. Buro, "Adversarial Hierarchical-Task Network Planning for Complex Real-Time Games," in *International Joint Conference on Artificial Intelligence (IJCAI)*, 2015, pp. 1652–1658.

[8] N. A. Barriga, M. Stanescu, and M. Buro, "Game Tree Search Based on Non-Deterministic Action Scripts in Real-Time Strategy Games," *IEEE Transactions on Computational Intelligence and AI in Games (TCIAIG)*, 2017.

[9] N. Justesen and S. Risi, "Learning Macromanagement in StarCraft from Replays using Deep Learning," in *IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2017, pp. 162–169.

[10] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998, vol. 1.

[11] S. Wender and I. Watson, "Applying Reinforcement Learning to Small Scale Combat in the Real-Time Strategy Game StarCraft:Broodwar," in *IEEE Conference on Computational Intelligence and Games (CIG)*, 2012, pp. 402–408.

[12] N. Usunier, G. Synnaeve, Z. Lin, and S. Chintala, "Episodic Exploration for Deep Deterministic Policies: An Application to StarCraft Micromanagement Tasks," Tech. Rep., 2016.

[13] P. Peng, Q. Yuan, Y. Wen, Y. Yang, Z. Tang, H. Long, J. Wang, and A. Group, "Multiagent Bidirectionally-Coordinated Nets for Learning to Play StarCraft Combat Games," Tech. Rep., 2017.

[14] R. S. Sutton, D. Precup, and S. Singh, "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning," *Artificial Intelligence*, vol. 112, pp. 181–211, 1999.

[15] S. Ontanón, "The combinatorial multi-armed bandit problem and its application to real-time strategy games," in *AAAI Conference on Artificial Intelligence in Interactive Digital Entertainment (AIIDE)*, 2013.

[16] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3, pp. 279–292, 1992.

[17] M. Preuss, D. Kozakowski, J. Hagelback, and H. Trautmann, "Reactive strategy choice in StarCraft by means of Fuzzy Control," in *IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2013, pp. 1–8.

[18] D. W. Aha, M. Molineaux, and M. Ponsen, "Learning to win: Case-based plan selection in a real-time strategy game," in *Case-based reasoning research and development*. Springer, 2005, pp. 5–20.

[19] A. Tavares, H. Azpúrua, A. Santos, and L. Chaimowicz, "Rock, Paper, StarCraft: Strategy Selection in Real-Time Strategy Games," in *AAAI Conference on Artificial Intelligence in Interactive Digital Entertainment (AIIDE)*, 2016, pp. 93–99.

[20] F. Sailer, M. Buro, and M. Lanctot, "Adversarial planning through strategy simulation," in *IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2007, pp. 80–87.

[21] S. Ontanón, G. Synnaeve, A. Uriarte, F. Richoux, D. Churchill, and M. Preuss, "A survey of real-time strategy game AI research and competition in StarCraft," *IEEE Transactions on Computational Intelligence and AI in Games (TCIAIG)*, vol. 5, no. 4, pp. 293–311, 2013.

[22] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *Journal of Artificial Intelligence Research*, 2012.

[23] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[24] T. D. Kulkarni, K. Narasimhan, A. Saeedi, and J. Tenenbaum, "Hierarchical Deep Reinforcement Learning: Integrating Temporal Abstraction and Intrinsic Motivation," in *Neural Information Processing Systems (NIPS)*, 2016, pp. 3675–3683.

[25] M. C. Machado, M. G. Bellemare, and M. H. Bowling, "A Laplacian Framework for Option Discovery in Reinforcement Learning," in *International Conference on Machine Learning (ICML)*, 2017, pp. 2295–2304.

[26] M. L. Littman, "Markov games as a framework for multi-agent reinforcement learning," in *International Conference on Machine Learning (ICML)*. New Brunswick, NJ: Morgan Kaufmann, 1994, pp. 157–163.

[27] L. Li, T. J. Walsh, and M. L. Littman, "Towards a Unified Theory of State Abstraction for MDPs," in *Proceedings of the Ninth International Symposium on Artificial Intelligence and Mathematics*, 2006, pp. 531–539.

[28] D. Churchill, A. Saffidine, and M. Buro, "Fast Heuristic Search for RTS Game Combat Scenarios," in *AAAI Conference on Artificial Intelligence in Interactive Digital Entertainment (AIIDE)*, 2012.

[29] A. R. Tavares, S. Anbalagan, L. S. Marcolino, and L. Chaimowicz, "Algorithms or actions?: A study in large-scale reinforcement learning," in *International Joint Conference on Artificial Intelligence (IJCAI)*, 2018.

[30] P. McCracken and M. Bowling, "Safe strategies for agent modelling in games," *AAAI Fall Symposium on Artificial Multi-agent Learning*, pp. 103–110, 2004. [Online]. Available: http://www.aaai.org/Papers/Symposia/Fall/2004/FS-04-02/FS04-02-014.pdf

[31] B. G. Weber and M. Mateas, "A data mining approach to strategy prediction," in *IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2009, pp. 140–147.

[32] G. Tesauro, "Temporal difference learning and TD-Gammon," *Communications of the ACM*, vol. 38, no. 3, pp. 58–68, 1995.

[33] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[34] A. Rosenfeld, M. E. Taylor, and S. Kraus, "Speeding up Tabular Reinforcement Learning Using State-Action Similarities," in *Adaptive Learning Agents Workshop at AAMAS*, 2017.

Generating Novice Heuristics for Post-Flop Poker

Fernando de Mesentier Silva
New York University
Game Innovation Lab
Brooklyn, NY
fernandomsilva@nyu.edu

Julian Togelius
New York University
Game Innovation Lab
Brooklyn, NY
julian@togelius.com

Frank Lantz
New York University
Game Center
Brooklyn, NY
frank.lantz@nyu.edu

Andy Nealen
New York University
Game Innovation Lab
Brooklyn, NY
nealen@nyu.edu

***Abstract*—Agents now exist that can play Texas Hold'em Poker at a very high level, and simplified versions of the game have been solved. However, this does not directly translate to learning heuristics humans can use to play the game. We address the problem of learning chains of human-learnable heuristics for playing heads-up limit Texas Hold'em, focusing on the post-flop stages of the game. By restricting the policy space to fast and frugal trees, which are sequences of if-then-else rules, we can learn such heuristics using several methods including genetic programming. This work builds on our previous work on learning such heuristic rule set for Blackjack and pre-flop Texas Hold'em, but introduces a richer language for heuristics.**

***Index Terms*—beginner heuristics, genetic algorithms, poker**

I. Introduction

If you were teaching someone to play Poker, but could only tell them a single rule for how to play, what would that rule be? That you should raise if you have a pair but fold otherwise? Now, if you could also tell them a second rule, what would that be?

Every journey starts with a first step and learning a complex skill starts with learning a simple part of that skill. Learning to drive a car usually starts with learning to shift gears, or in the US, to find the brake. This is no different in games. Playing Super Mario Bros, you will first learn to move sideways, then to jump over enemies, then to jump on top of enemies, bump into question mark blocks, shoot fireballs etc. Much later in the game you will learn complex combinations of these such as wall-jumping or finding secret passages. Every simple thing you learn makes you play the game better. These units of learning have been called "skill atoms" or "heuristics"; in this paper, we'll use the latter.

Being able to automatically subdivide a complex skill into heuristics would be very useful for being able to automatically (or semi-automatically) generate instructional sequences or tutorials for that skill. Given that we know what heuristics must be learned, and in what order, the task of figuring out how to teach those heuristics becomes much easier. But finding the heuristics necessary for a task is important for at least one other reason as well: to gauge the depth of the task. Deep tasks, for example deep games, are characterized by that mastering them requires learning a long chain of heuristics that build on each other. This can be contrasted with shallow tasks/games where there are only a small number of effective heuristics available, or there is a large number but they cannot be used together. Automatically finding the heuristics can therefore also help us estimate the depth of the task.

In this paper, we investigate methods for finding heuristics for post-flop heads-up limit Texas Hold'em Poker. This is one of the simplest versions of Poker available (chosen because it is somewhat tractable) and we are only tackling the later stages of the game. This follows on from our earlier work on finding heuristics for Blackjack and for the pre-flop phase of the same type of Poker. The basic idea is to use and search in the space of heuristic chains, and evaluate them by playing against a strong adversary.

Like in our previous work, we apply and compare greedy exhaustive search, axis-aligned search, and genetic programming, and represent heuristics in a domain-specific language meant to be able to express human-learnable rules. A major contribution in this paper with respect to our previous work is a richer language for the heuristics, able to capture late-game aspects of Poker. Other novel contributions include our approach to finding initial positions for the gameplay simulations, and our table-based reduced adversarial agent which allows fast simulation.

In the next section we describe previous work on heuristics, genetic programming, Poker AI and other topics. We then describe the particular Poker variant we are addressing, and the adversarial agent we employ. Next, we describe the heuristic language that we developed for post-flop Poker, and the details of the algorithmic approaches. In the results section, we perform comparative analysis of the quality of heuristics found through the different approaches and plot curves of policy strength relative the the length of the heuristic chain, as well as show examples of particular heuristics found.

II. Background

When making decisions, novices are not the only ones that use simple models. The theory of bounded rationality claims that the human decision making process is a factor of the amount of time available to make a decision and the amount of information available [1], [2]. Opting for simple rules-of-thumb can provide better outcome when compared to using complex algorithms because they are less prone to errors in the execution [3].

Agents that target optimum play have been able to show impressive results, in some cases being able to compete or

978-1-5386-4359-4/18/$31.00 ©2018 IEEE

surpass professional human players. Approaches for Checkers [4], Chess [5], Go [6], Othello [7] and Poker [8] are some of the examples of such agents. Unfortunately, most of these solutions require large memory space or heavy computation that are unfeasible to be executed by human players. In order to create powerful strategies, players use heuristics to approximate the evaluation of a complex scenario. Our paper contrasts with approaches that look to maximize quality of the solution in that we constrain our heuristics to be simple to understand and execute, being careful to analyze the relationship between quality and complexity. Furthermore, we target novice players who would have been just introduced into the game.

Evolutionary approaches have also been successful at generating high performance agents. Players have been evolved for adversarial games such as Poker [9], Backgammon [10] and Lose Checkers [11]. Evolutionarhy approaches for developing strategies have also been applied to a clone of Super Mario Bros where neural nets were evolved to create controllers that play the game [12], to the solo variant of Pong [13], and to evolve controllers that exhibit general driving skills [14].

Good game-playing in poker has many different aspects to it. Bluffing, reading your opponent and calculating the risks involved in the betting are essential for high skill play [15], [16], [17]. Another important aspect of the game are what is called mixed strategies. With a mixed strategy, when observing the same game state, action A is played with probability X and action B with probability Y, as opposed to a deterministic decision. The heuristics generated in this paper are deterministic, resembling rules-of-thumb, which are known to only have a reasonable performance at the beginner level [17]. That don't pose a problem to us since beginner players are exactly the ones we wish to target with our heuristics.

The work on this paper follows our previous work on generating simple heuristics for Blackjack [18] and for the Pre-Flop round of Heads-Up Limit Texas Hold'em [19]. Using Blackjack we were able to show that it is possible to generate these heuristics and by evaluating their performance versus their complexity we approximated the skill chain [20] of the game. Blackjack was a good first step since it is a 1.5 player game, the opponent is the dealer that always plays following a known deterministic algorithm. In the Pre-Flop round paper, we showed that the same algorithms were indeed capable of generating simple heuristics on a 2-player game. In a multiplayer game your opponents are able to react and reshape their strategies during gameplay to adapt to yours, that brings the question of whether the heuristics we generated were subjected to a intransitivity relationship, meaning heuristic A beats heuristic B, heuristic B beats heuristic C, but C is able to beat A. We showed that there was in fact intransitvity between our heuristics. In this work we expand the analysis to the Post-Flop rounds of Poker, which has a richer vocabulary due to more information being available in the game state. Furthermore, we propose how a player with knowledge of the heuristics of all rounds could be able to use those to play a complete match of Heads-Up Limit Texas Hold'em.

III. Heads Up Limit Texas Hold'em Poker

Poker is a popular gambling card game. Having many variants and being played all around the world, whether it be online, casually at casinos or at a professional level. One of the most popular variations of Poker is Texas Hold'em. In this work, we will discuss how to generate novice-level simple heuristics for playing this variant of the game.

In Texas Hold'em, 2 or more players bet over multiple rounds on the best 5 card poker hand they can make out of the cards in play. The game has stochastic elements, with cards being distributed from a randomly shuffled deck, hidden information with the players holding cards in secret from each other, and as gameplay progresses more information is added, with cards that are shared by all players being revealed and actions being made. Strategies in the game usually involve bluffing, reading your opponents and assessing the risks of performing each action. Good strategies tend to have a lot of moving parts that are usually overwhelming for beginners.

A match of Texas Hold'em has 4 card rounds (in this respective order): Pre-Flop, Flop, Turn and River. Each of these rounds consists of card dealing followed by betting. In the Pre-Flop round, players are each dealt 2 cards they keep in their hand, secret from the others. In the Flop round, 3 cards are dealt face-up on the table, these are cards that are shared between all players. In both the Turn and River 1 more card is dealt to the pool of shared cards, totaling 5 face-up cards at the end. After the last round of betting, if there is more than 1 player still in the match, the showdown happens. In the showdown, the players remaining form the best 5 card poker hand they can from a total of 7 cards, their 2 cards from the Pre-Flop and the 5 cards face-up on the table. The player with the most valuable hand, following the standard hand ranking comparison, wins the total pot. On this paper, we focus on generating heuristics for playing the betting rounds in the Flop, the Turn and the River, following our previous work on the Pre-Flop round [19].

During betting, players alternate turns choosing to take 1 of the 3 possible actions. The action *Fold* has the player discarding their hand and being out of the game until the end of the match, meaning that player will not participate in the showdown. The *Check/Call* action has the player matching the largest bid made by another player so far. When using the action *Raise* players increase the highest bet to this point, turning them into the highest bidder. Once play comes back to the last player to have *Raised*, if there are multiple players remaining, the betting round ends and the match proceeds to the next round.

For the scope of this work we will be using the variant Heads-Up Limit Texas Hold'em. Heads-Up means that the matches are played by only 2 players. The variant Limit means that every time players *Raise* they can only do it by a fixed amount. Furthermore, we utilize the same settings as the Annual Computer Poker Competition for Heads-Up Limit: Only a total of 3 *Raises* can happen at Pre-Flop and 4 in the other rounds. After the limit of *Raises* has been met, the player

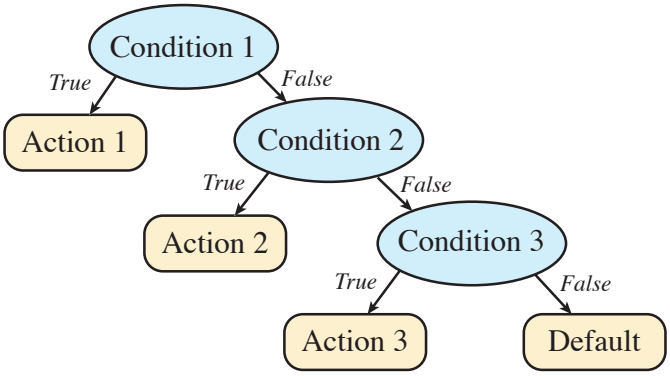


Fig. 1. Structure of a Fast and Frugal Tree (FFT). In this binary tree, each node in blue represents a condition and each node in yellow represents an action. When a condition is satisfied, the FFT returns the left child of that condition, otherwise it proceeds to next condition or, if it reached the end, returns the default action.

has to choose a different action.

The decision to use this variant of Poker is to have a simpler game to build our analysis on, and still have significant result, considering that a lot of the principles behind good heuristics in Heads Up Limit Texas Hold'em can be used to play other variants. Those are some of the reasons why this specific variant has been the focus of a lot of research, culminating in an agent that weakly solved this game [21].

IV. Heuristic Structure

Since we are looking to generate heuristics that are simple and effective, we need a proper representation. Commonly used in bounded rationality theory, Fast and Frugal Tree (FFT) is a type of binary search tree. In FFTs, nodes that have children represent conditions that inform decisions, while the leaves represent the action to be executed following the parent's outcome. Figure 1 shows the structure of FFTs. In order to make a decision we start at the condition in the root. If that condition is satisfied, we return the action on the left child of this node. If that condition fails, we proceed to the right child condition node. In case there are no more conditions to be tested, the default action (right child of the condition node of highest depth) will be returned. FFTs can also be expressed as decision lists [22], [23] and commonly have a higher performance, in practice, when compared to more complex algorithms [24].

Throughout the following sections of this paper we will represent FFTs as a chain of if-then-else statements. Their structure is as follows:

if Condition 1 **then** Action 1
else if Condition 2 **then** Action 2
else if ... **then** ...
else Default Action

We will refer to the grouping of a *condition* and the *action* resulting from successfully meeting such condition as *statement*. Each *condition* is formed by one or more boolean tests, called *clause*, that satisfy the *condition* when, and only when, are all evaluated to true. This means that a *condition* is formed by a conjunction of *clauses*. Lastly, the *action* returned by reaching the else statement is therefore the *default action*.

A. Poker Clauses

In order to generate heuristics, we first need to define a vocabulary to express them. The vocabulary is the basis of all heuristics, and determines what kind of information about the game state can be used. It is composed of several boolean functions that can be called as clauses for a condition.

When comparing to our previous work on generating heuristics for the Pre-Flop round of Heads-Up Limit Texas Hold'em, one of the main differences is the vocabulary being used. Since during the Pre-Flop players only have access to their two cards, a consequence is that the clauses rely heavily on those. For the rounds after, we no longer address the two cards in hand specifically, but rather the game it is forming together with the shared face-up cards. The only aspect of the game state that is common across all rounds is the size of the pot.

The vocabulary used in this work was built following common points of analysis discussed by players and frequently analyzed in books about Poker, such as *Play Poker Like the Pros* [15] and *The Mathematics of Poker* [16].

In the remainder of this section we describe the different boolean functions and clauses used to generate the heuristics.

1) ***Hand Rank****:* We define *Hand Rank* as the value the current player hand would have at the showdown. Since there is an order for the quality of different hands, it is possible to make statements such as: $FullHouse > Pair$ and $Flush < FourOfAKind$. Given such, we analyze *Hand Rank* in the interval: $HighestCard \leq HandRank \leq RoyalStraightFlush$. It is worth noting that this does not take into account the value of the highest card in the hand, used to break ties between hands with the same *Hand Rank*. For such, we have another boolean function.

2) ***Highest Valued Card in Hand****:* When there is a tie between 2 hands of the same *Hand Rank*, the value of the highest card that composes that hand is used to break ties. For hands of $ThreeOfAKind$ or higher it is not uncommon to play them through even when the value of the card used to form it is low. That is not the case for the lower ranks though. Considering that the bottom 3 *Hand Ranks* happen more often then all others, testing the value of the tie breaker is due necessary. Given the range of cards, the interval that can be analyzed in a clause is: $2 \leq x \leq 14$.

3) ***Total Pot****:* The size of the pot is important when playing the game since it represents the rewards for winning that match. In the heuristics we show, we represent *Total Pot* in relation to how many Big Blinds it represents. So, a test such as $TotalPot \geq 30$ is true when the pot is equal or greater than 30 Big Blinds. In Poker, Big Blind refers to the amount the last player to act in the Pre-Flop is forced to bet in the beginning of the game, to account for their favorable position.

4) ***Aggressive Opponent****:* Evaluating the behavior of the opponent can give valuable information about how a player is to proceed with their actions. Since building and analyzing

a model for the adversary is beyond the scope of this paper, we have a simplified but significant representation. We add to the vocabulary 2 boolean functions: $IsOpponentAggressive$ and $NotIsOpponentAggressive$. The first is satisfied when the last move the opponent made was *Raise*. The second just returns the inverse of the first' result.

5) ***Texture of the Board****:* By observing the face-up cards present in the board, we can analyze their texture. Texture of the Board indicates whether the shared cards shown make it likely or unlikely for players to build a good hand. For such, we create $IsBoardDry$, that is satisfied when the board is dry (unlikely to form good hands with) and $IsBoardWet$ which represents the opposite.

6) ***Outs****:* A very common and powerful strategy in Texas Hold'em is to be aware of the number of outs you have. The number of outs represents the number of cards left in the deck that if drawn to the board at later stages can improve your *Hand Rank* from what it currently is. We count the number of cards to turn Pairs into Three of a Kind, number of cards that can turn a 2 pair into a Full House and number of cards that can complete the a Flush or Straight missing 1 card. This function is not used for the River round, since no more cards will be added to the board.

V. Opponent Agent

In order to find good simple heuristics, we need to search the space of possible FFTs and evaluate their quality. To estimate the quality of a heuristic, which we will call fitness for this point on, we rely on playing a substantial amount of games and evaluating the average outcome.

Since Heads-Up Limit Texas Hold'em is a 2-player game, the fitness is calculated in relation to playing the different heuristics against the same opponent. In order for it to be feasible to play enough games in a short amount of time, we require an automated agent to play against.

For our work on Pre-flop, we were able to use the table that is part of the Nash Equilibrium used to solve the game [21]. For the post-flop rounds this table would be too large, and we don't require optimal play. Therefore we resort to a method similar to the one used to weakly solve the game. Another caveat is the time it takes for the agent to be trained. These motivate using the same methods, but under a simplified strategy, while making it more vulnerable.

The Opponent Agent we created is inspired by the work done towards building competitive AI for Heads-Up Limit Texas Hold'em [25], [26], [27] and the agent that was able to weakly solve the game [21].

This agent is built using the Counterfactual Regret Minimization (CFR) algorithm. The agent, as opposed to our heuristics which are deterministic, plays a mixed-strategy, meaning that depending on the game state it has different probabilities of making moves X, Y or Z. By starting from an equal probability for making any of the 3 moves, the algorithm then plays against itself and updates its probabilities based on the amount of regret of having performed each action. With the regret values, it updates the probabilities in relation to their positive regret (how much better it would have done if it picked another action). The algorithm has to be trained over an expressive amount of runs, relative to the size of the space that is being represented.

Since computing individual probabilities for each possible card combination is beyond the scope of this work (and tackled in Bowling *et al.* [21] and Tammelin *et al.* [28]), we opt to use a very simple card representation abstraction. For every 1 of the possible 57,344 sequences of actions, the agent classifies their *Hand Rank*, dividing hands of different rank into different buckets. This creates a probability distribution that reflect knowledge of the game and that is feasible to be trained in a small amount of time. That said, the agent has very small granularity, meaning it is vulnerable to being exploited. But, since it represents a more general play, it is unlikely to bias the heuristics generated against it. Our agent was trained using 30 cores for 7 hours, having played over 5.5 billion matches.

VI. Algorithms

Once we have an Opponent Agent, we can define our Fitness function. Such will be the average amount of money earned from playing against the agent for 400,000 matches. And the end of every match we reset each players pot and we guarantee that our heuristic will be first player through half the matches and second player for the others. Subjects with higher fitness are ranked higher in the algorithms.

We now introduce the algorithms we use to generate simple heuristics. Increasing the fitness usually reflects in an increase in complexity. So, in order to form a diverse population and be able to approximate the skill chain [20], we target creating heuristics of growing complexities, such as we did with the game Blackjack [18].

Since we are generating heuristics for the different rounds in the game, we run the algorithms separately for each case. Since the game develops over previous rounds, to avoid bias, we have the Opponent Agent play against itself until it reaches the start of the round we want to generate heuristics for. From there, we have the game be played by the heuristic until the end of the current round. Once the round is over, we deal out any cards that haven't been dealt yet, as if there was no more betting rounds in the game. After such, we proceed to the showdown to decide the outcome of this match. Once we generate heuristics for each stage of the game we will analyze how we can use these to create a strategy for playing a match from start to finish, using only heuristics found.

A. Greedy Exhaustive Search

With this algorithm, we start by initializing the population with all possible FFTs that are composed by a single condition with a single clause. They are created by assigning the boolean functions to clauses using all possible values in range and also varying all actions to be any of the 3 possible.

We proceed to evaluating the fitness of all individuals in the population, and the one with highest fitness is then the best 1-statement FFT possible. If we desire to have an individual with more statements, we repeat the process of generating all

1-statement possibilities and appending these to the end of the heuristic found on the previous step.

Since as the number of statements grow the amount of possible FFT we can build increases exponentially, it is unfeasible to exhaustively explore all possible heuristics with more than 1-statement. Even by just allowing for a statement to have more than 1 clause, the space of possible heuristics grows exponentially. As a consequence, the algorithm is very prone to finding local maximum as opposed to approaching the global optimum. Even with these caveats, it is still useful for finding the heuristics with the least amount of complexity.

B. Axis-Aligned Search

Since the space of possible heuristics is very large, performing a search for quality heuristics is computationally expensive. To avoid such, we created the Axis-Aligned search.

The method is inspired by computer graphics techniques, in particular line search, and it targets optimizing individual dimensions, one at a time, rather than the whole. The algorithm starts from a randomly generated heuristic. Then it creates a copy of the FFT, for each clause and action in the heuristic, changing only the value being compared in a clause or which action is returned. A copy is created for every possible value that can be tested for every possible clause. Copies are also created for every action. We now have a population of all these small variants of the original heuristic. We calculate the fitness for all of individuals and find the one with the highest value. The most fit now becomes the main subject. We then repeat the process, now varying all clauses and actions, except that we lock the clause/action that was changed from further mutating for the rest of the iterations. We continue with iterations until all clauses and actions are locked, or when the most fit individual of an iteration is the initial subject.

The quality of the results from this method are reliant on the random heuristic generated at the beginning. However, Axis-Aligned Search runs faster than the Genetic Algorithm, so we can have multiple runs and pick the most fit individual out of all of them. The algorithm can also be used to optimize the results found through other methods, by seeding a generated heuristic to the first iteration, instead of randomly creating one.

C. Genetic Programming

The most robust and the one most successful method at finding close to optimum heuristics for different complexities. Usually the best heuristics found come from using Genetic Programming [29], [30], [31] to search the space.

The algorithm starts with a population of 100 FFTs, of a fixed number of statements, generated at random. Then, a generation is executed by evaluating the fitness of all individuals of the population. The top 50, the elite, are selected to move to the next generation. The bottom 50 are discarded. New individuals are generated: 20 by mutating copies of elite individuals, 30 by randomly crossing over copies of the elite and then mutating the children. By maintain the top half of the population we assure that poor performing genomes do not propagate, while keeping the population diverse through having a large size for the elite and performing mutation and crossover. The algorithm performs 100 generations.

if $HandRank \geq Pair$ **then** RAISE
else CHECK/CALL

Fig. 2. Most fit least complex heuristic for Flop round. Found by Exhaustive Search, fitness is approximately 0.89.

When mutating a heuristic, the algorithm visits every clause, every constant in a clause and every action and mutates it with a 30% probability. When a clause is mutated, it is replaced with an entirely new one. When a constant is mutated, it is replace with a new constant that falls within the interval accepted by that clause. When an action is mutated, one of the other 2 actions substitutes it. Heuristics that return the same action for both evaluations of the last condition and heuristics that have a condition that always evaluates to the same value are not accepted into the population, neither are duplicates of other heuristics already in the population, and the mutation is repeated until a valid individual is created.

When crossing over 2 heuristics, a random condition or action is selected and it crosses over with the condition or action at the same depth/position in the other FFT. As it is with the mutation, with crossover creates a invalid individual or a copy of an individual already in the population, the step is repeated until a pair of valid children are returned.

Despite being the best at finding close to optimum individuals, the genetic algorithm is the most computationally expensive being roughly 25 times longer than 1 run of Axis-Aligned Search. As it is going to be discussed in the next section, the Genetic Algorithm found the majority of the most fit individuals we have for each complexity.

VII. RESULTS

In this section we showcase the results of running the algorithms described previously to generate simple heuristics for the Flop, Turn and River rounds of Heads-Up Limit Texas Hold'em. For each round we demonstrate 2 heuristics: the most fit of the lowest complexity and the most fit overall.

In order the evaluate the results found, we measure the complexity of a heuristic. For the remainder of the paper, complexity will refer to the sum of the number of clauses and number of actions (including the default action) of a heuristic. Although this evaluation does not take into account that some clauses or statements might be harder than others to parse for humans, it provides a good approximation of the amount of information that is represented.

A. Flop

Figure 2 shows the lowest complexity most fit heuristic for the Flop round. This heuristic raises if it is currently holding a hand with a pair or higher and checks/calls otherwise. Since the heuristic was trained in games that end after the Flop, holding a pair at this stage of the game can be a powerful hand, but one that can lose strength if play was to move past this round. Since there are still 2 cards left to be dealt to the

if $HandRank \geq Pair$ **then** RAISE
else if $TotalPot \leq 2$ **then** RAISE
else if $Outs \geq 11$ **then** RAISE
else CHECK/CALL

Fig. 3. Most fit heuristic for Flop round. It has a complexity of 7. Found by Genetic Programming, fitness is approximately 1.28.

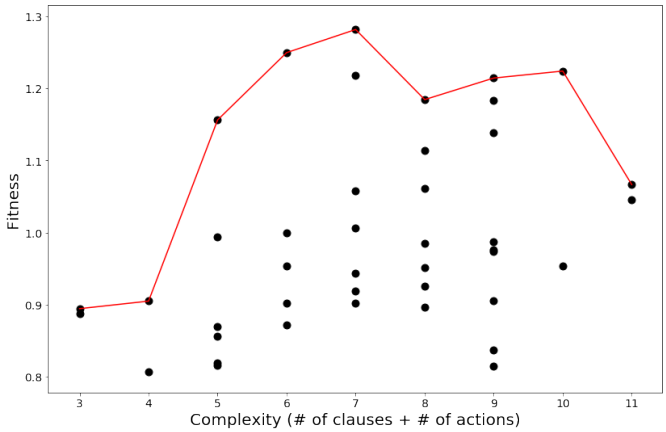


Fig. 4. Plot showcasing the relation between fitness and complexity for the heuristics found for the Flop round. The Y-axis represents fitness and X-axis represents complexity. Each dot represents 1 heuristic. The red line is an approximation of the skill chain found from the heuristics generated. Heuristics found with fitness bellow 0.8 were omitted from the graph.

board, a pair would beat other hands that have a higher reward expectation in the long run, such as a hand that is 1 card away from making a Flush.

This heuristic having positive fitness (as observed with other heuristics shown for Flop and River) indicates that it beat the opponent agent over time, but since play is cut short and later betting is skipped, it cannot be pointed as an indication that this heuristics dominates the opponents strategy.

In contrast, Figure 3 demonstrates the most fit heuristic, of complexity 7, found for the Flop round. The heuristic raises on any hand that has a pair or higher. The second statement represents a very specific scenario, the only sequence of actions that reach the Flop with total pot being 2 or lower is when all players have only checked/called in the Pre-Flop. The third statement clause checks for a state that is very advantageous to the player. If the player has 11 or more outs during the Flop, considering that at this round the deck has 45 cards and 2 more cards will be added to the board, it means that approximately 25% of the deck will improve the player's hand. Raising the pot is a strong action in these conditions.

Worth noticing is that both heuristics shown share the same first statement. This means that it could be possible to reach the best heuristic found by iterating on the best simplest heuristic. However, it is unlikely that the Greedy Exhaustive Search would have reached this result, specially considering that the second statement covers a very small part of the space of possible game states.

Figure 4 displays the relationship between fitness and complexity for the heuristics we generated. We can observe a fitness increase from complexity 3 through 7. While moving from complexity 3 to 4 grants a small gain in fitness, the greatest gain comes from reaching complexity 5. This can be explained by observing that while heuristics of complexity 3 and 4 have 1 statement, the best of complexity 5 has 1 more.

It is also worth noting that the peak of the graph is present on complexity 7, despite there being plenty of samples for complexity 8 and 9. It is unlikely that there are no heuristics of higher complexity that can outperform the current peak. It is also easy to construct heuristics of higher complexity that simply imitate the functionality of lower-complexity heuristics. It is very likely that additional search time will allow to us find heuristics with higher complexity of at least equivalent fitness. We can extend this concept to explain the sharp drop from complexity 10 to 11.

if $HandRank \geq TwoPair$ **then** RAISE
else CHECK/CALL

Fig. 5. Most fit least complex heuristic for Turn round. Found by Exhaustive Search, fitness is approximately -0.61.

if $HandRank \leq HighestCard$ **then** CHECK/CALL
else if $isBoardDry$ **then** RAISE
else if $totalPot \geq 31$ **then** CHECK/CALL
else if $HandRank \leq Pair$ **then** CHECK/CALL
else RAISE

Fig. 6. Most fit heuristic for Turn round. It has a complexity of 9. Found by Genetic Programming, fitness is approximately 0.56.

B. Turn

When observing the most fit simplest heuristic for the Turn, showcased in Figure 5, we can notice a similarity with the heuristic for the Flop. Changing from raise on pair or higher to raise on two pair or higher models a more conservative playstyle. Furthermore, it is noticeable that the fitness that before was positive is now negative. It indicates the opponent agent becomes harder to exploit further into the game.

The most fit heuristic for the Turn, represented on Figure 6, is the most granular of the ones detailed in this paper. The first statement has the players checking/calling if they reach the Turn and don't even have a pair. Next, it evaluates if the board is dry, and if so chooses to raise. This means it raises if it is holding a pair or higher and the board is dry, which puts the player in a position that seems advantageous. It proceeds to check/call if the total pot is greater or equal to 31 Big Blinds, which would indicate that the opponent has been constantly investing in their hand. In the case that the pot isn't as great, and the board is wet, it chooses to be slightly conservative and check/call if the best it has is a pair, otherwise it will raise. The heuristic has a positive fitness, meaning it is able to exploit the opponent in a more advanced stage of the game, but only by half the margin that we have reported for the Flop.

When comparing the plot for Turn, shown on Figure 7, and the Flop plot, we notice 2 main differences. The first being the

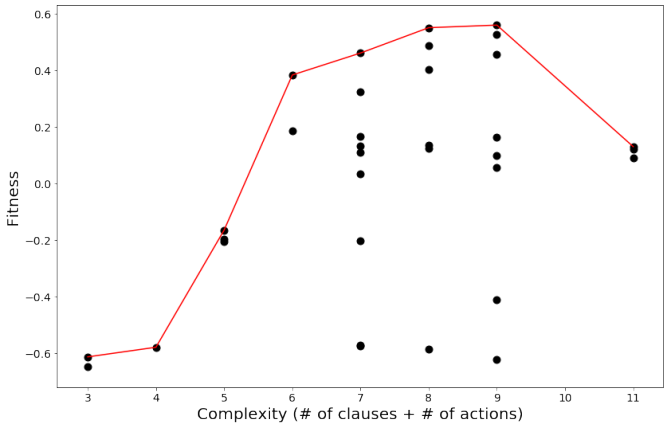


Fig. 7. Plot showcasing the relation between fitness and complexity for the heuristics found for the Turn round. The Y-axis represents fitness and X-axis represents complexity. Each dot represents 1 heuristic. The red line is an approximation of the skill chain found from the heuristics generated. Heuristics found with fitness bellow -0.7 were omitted from the graph.

if $HandRank \leq Pair$ **then** CHECK/CALL
else RAISE

Fig. 8. Most fit least complex heuristic for River round. Found by Exhaustive Search, fitness is approximately -2.27. This heuristic is analogous to the one shown in figure 2.

sharp increase that was from complexity 4 to 5 repeats, but is followed by another sharp increase from 5 to 6. The other is that there is a gain from incrementing complexity from 3 all the way to 9. It is possible that since the Turn states have more information, an extra card on the board, than the Flop it reflects on the space of viable good heuristics, making it easier for an algorithm such as Genetic Programming to find good individuals. Another observation that is worth making is how the fitness interval shifted, going from -0.6 to 0.6, when compared to the all positive interval of the Flop graph.

C. River

The most fit least complex heuristic for the last round of the game, the River, show on Figure 8, is the exact same found for the Flop and shown back on Figure 2. Despite playing exactly the same, the fitness is considerably smaller, going from 0.89 in the Flop to -2.27 in the River. This is likely due to the opponent agent having had the chance to play the game from start to finish, making it considerably more efficient than when the future betting rounds were skipped. Another detail to notice is how the best complexity 3 heuristic became more conservative only for the Turn, that is due to there is still being a lot of opportunities for improvement after the Flop and that the potential positive return of checking/calling with a pair or lower outweigh the negative return of folding at this stage.

The most fit heuristic for the River, shown on Figure 9, has 2 unique features when compared to all the others we analyzed so far: it is the only heuristic that folds and is the only heuristic to observe the opponents last action. The heuristic starts by evaluating if it the best hand it has is only a highest card and in

if $HandRank \leq HighestCard$ **AND** $OppAggressive$
then FOLD
else if $HandRank \leq TwoPair$ **then** CHECK/CALL
else if $totalPot \geq 26$ **then** FOLD
else RAISE

Fig. 9. Most fit heuristic for River round. It has a complexity of 8. Found by Genetic Programming, fitness is approximately -0.41.

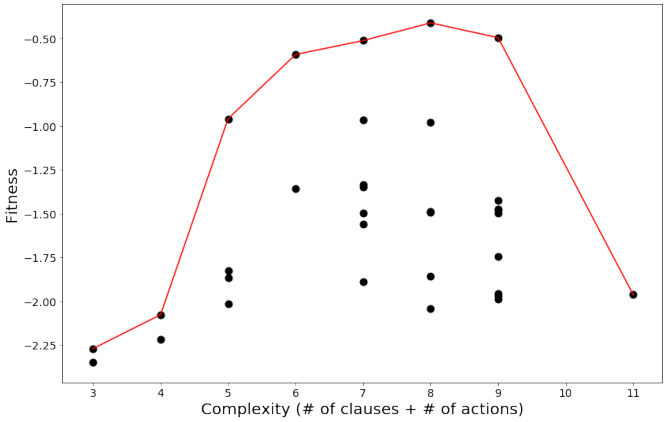


Fig. 10. Plot showcasing the relation between fitness and complexity for the heuristics found for the River round. The Y-axis represents fitness and X-axis represents complexity. Each dot represents 1 heuristic. The red line is an approximation of the skill chain found from the heuristics generated. Heuristics found with fitness bellow -2.5 were omitted from the graph.

case that is true and the opponent is being aggressive with their last move. In that case it decides to fold. It is counterintuitive that folding is only part of any heuristic when we reach the last round of the game. It proceeds to Check/Call if it is holding two pairs or lower. Lastly it once again folds if the pot is above 25 Big Blinds, otherwise it raises.

By analyzing the graph for River heuristics, shown on Figure 10, we observe that it fits somewhere in between the 2 previous. There is only 1 sharp increase, from complexity 4 to 5, but it peaks at complexity 8, with complexity 9 being considerably close in fitness. The unique feature of this graph is that the fitness interval is completely negative, meaning no matter which of the heuristics found we play, we'll lose money to the opponent agent. This corroborates our hypothesis that, when it is able to play the full game plan, from start to finish, the Opponent Agent's strategy becomes less exploitable than when skipping later betting on previous rounds.

D. Playing the full game with heuristics

With the results from our work on Pre-Flop heuristics [19] and the ones just presented, we raise the question: How well can we play the game using only our heuristics? A complete exploration and discussion of how to approach this is out of the scope of this work, but we are interested in briefly discussing the most naive approach to this: Play each round of the game with the most fit heuristic we have found for that round.

Using this method, we played 400,000 games between our best heuristics (the single best heuristic for each phase) and

our opponent agent. To our slight surprise, the set of heuristics outperformed the opponent agent, with a fitness of 3.43. Our best explanation for this is that the heuristics, which have only been trained on this particular agent, have overfit to that agent and learned to exploit it. An alternative explanation is that the representation used to create the opponent agent is too limited, and that the heuristic representation, despite its apparent simplicity, is capable of learning powerful strategies.

Even though our multiple heuristic player performs well against the Opponent Agent, it remains much more vulnerable. This is due to the fact that our heuristics are deterministic; if the opponents are able to read the heuristics play style, they will be in a very good position to exploit it.

VIII. Conclusion

In this paper we presented techniques for generating simple novice-level heuristics for the Post-Flop rounds of Heads-Up Limit Texas Hold'em. We utilized 3 different algorithms to find such heuristics, with Genetic Programming being the most successful at generating the more complex heuristics and Greedy Exhaustive Search being the most practical for generating the simplest of 1-statement heuristics.

We then proceed to analyze the most fit simplest heuristic and the most fit of heuristics found for each of the 3 Texas Hold'em rounds being discussed. We observed and compared their differences and made an attempt at creating a parallel with the thought process behind the decisions the heuristics make. We also discussed the plots of fitness x complexity for the heuristics found for each round, and the curve that approximates the skill chain [20] for each. The graphs also helped observe the differences the games' design brings to each individual round of the game.

Lastly, we proposed a naive approach to build a strategy for the full game from only the heuristics we have found. By selecting the most fit heuristic of each round, we obtained a large positive reward against the same opponent the heuristics were trained in. This raises the question of if our heuristics learned to do better than expected, or whether the heuristics found were overfit to beat that specific opponent. The answers to these question are out of the scope of this work and will instead be proposed as future work.

Acknowledgment

Authors thank the support of CAPES, Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brazil.

References

[1] H. A. Simon, "Theories of bounded rationality," *Decision and organization*, vol. 1, no. 1, pp. 161–176, 1972.

[2] D. Kahneman, "Maps of bounded rationality: Psychology for behavioral economics," *The American economic review*, vol. 93, no. 5, pp. 1449–1475, 2003.

[3] G. Gigerenzer and D. G. Goldstein, "Reasoning the fast and frugal way: models of bounded rationality." *Psychological review*, vol. 103, no. 4, p. 650, 1996.

[4] J. Schaeffer, J. Culberson, N. Treloar, B. Knight, P. Lu, and D. Szafron, "A world championship caliber checkers program," *Artificial Intelligence*, vol. 53, no. 2, pp. 273–289, 1992.

[5] M. Campbell, A. J. Hoane, and F.-h. Hsu, "Deep blue," *Artificial intelligence*, vol. 134, no. 1, pp. 57–83, 2002.

[6] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[7] M. Buro, "From simple features to sophisticated evaluation functions," in *Computers and Games*. Springer, 1998, pp. 126–145.

[8] M. Moravčík, M. Schmid, N. Burch, V. Lisỳ, D. Morrill, N. Bard, T. Davis, K. Waugh, M. Johanson, and M. Bowling, "Deepstack: Expert-level artificial intelligence in heads-up no-limit poker," *Science*, vol. 356, no. 6337, pp. 508–513, 2017.

[9] L. Barone and L. While, "An adaptive learning model for simplified poker using evolutionary algorithms," in *Congress on Evolutionary Computation, CEC 99*, vol. 1. IEEE, 1999.

[10] Y. Azaria and M. Sipper, "Gp-gammon: Genetically programming backgammon players," *Genetic Programming and Evolvable Machines*, vol. 6, no. 3, pp. 283–300, 2005.

[11] A. Benbassat and M. Sipper, "Evolving lose-checkers players using genetic programming," in *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*. IEEE, 2010, pp. 30–37.

[12] J. Togelius, S. Karakovskiy, J. Koutník, and J. Schmidhuber, "Super mario evolution," in *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*. IEEE, 2009, pp. 156–161.

[13] W. B. Langdon and R. Poll, "Evolutionary solo pong players," in *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, vol. 3. IEEE, 2005, pp. 2621–2628.

[14] J. Togelius and S. M. Lucas, "Evolving robust and specialized car racing skills," in *IEEE Congress on Evolutionary Computation*. IEEE, 2006, pp. 1187–1194.

[15] P. Hellmuth, *Play poker like the pros*. Harper Paperbacks, 2003.

[16] B. Chen and J. Ankenman, *The mathematics of poker*. ConJelCo LLC, 2006.

[17] D. Sklansky, *The theory of poker*. Two Plus Two Publishing LLC, 1999.

[18] F. de Mesentier Silva, A. Isaksen, J. Togelius, and A. Nealen, "Generating heuristics for novice players," in *Computational Intelligence and Games (CIG), 2016 IEEE Conference on*. IEEE, 2016, pp. 1–8.

[19] F. de Mesentier Silva, J. Togelius, F. Lantz, and A. Nealen, "Generating beginner heuristics for simple texas holdem," *The Genetic and Evolutionary Computation Conference (GECCO)*, 2018.

[20] F. Lantz, A. Isaksen, A. Jaffe, A. Nealen, and J. Togelius, "Depth in strategic games," 2017.

[21] M. Bowling, N. Burch, M. Johanson, and O. Tammelin, "Heads-up limit hold'em poker is solved," *Science*, vol. 347, no. 6218, pp. 145–149, 2015.

[22] R. L. Rivest, "Learning decision lists," *Machine learning*, vol. 2, no. 3, pp. 229–246, 1987.

[23] D. Ashlock, M. Joenks, J. R. Koza, and W. Banzhaf, "Isac lists, a different representation for program," in *Genetic Programming 1998*. Morgan Kaufmann, 1998, pp. 3–10.

[24] G. Gigerenzer, "Fast and frugal heuristics: The tools of bounded rationality," *Blackwell handbook of judgment and decision making*, pp. 62–88, 2004.

[25] M. Zinkevich, M. Johanson, M. Bowling, and C. Piccione, "Regret minimization in games with incomplete information," in *Advances in neural information processing systems*, 2008, pp. 1729–1736.

[26] D. Billings, N. Burch, A. Davidson, R. Holte, J. Schaeffer, T. Schauenberg, and D. Szafron, "Approximating game-theoretic optimal strategies for full-scale poker," in *IJCAI*, 2003, pp. 661–668.

[27] A. Gilpin and T. Sandholm, "A competitive texas hold'em poker player via automated abstraction and real-time equilibrium computation," in *Proceedings of the National Conference on Artificial Intelligence*, vol. 21, no. 2. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2006, p. 1007.

[28] O. Tammelin, N. Burch, M. Johanson, and M. Bowling, "Solving heads-up limit texas hold'em," in *IJCAI*, 2015, pp. 645–652.

[29] J. R. Koza, "Genetic programming as a means for programming computers by natural selection," *Statistics and computing*, vol. 4, no. 2, pp. 87–112, 1994.

[30] O. Kramer, *Genetic algorithm essentials*. Springer, 2017, vol. 679.

[31] N. F. McPhee, R. Poli, and W. B. Langdon, "Field guide to genetic programming," 2008.

Scale-Free Evolutionary Level Generation

André Siqueira Ruela
Postgraduate Program in Information Systems
University of São Paulo
São Paulo, Brazil
andre.siqueira.ruela@gmail.com

Karina Valdivia Delgado
dept. of Information Systems
University of São Paulo
São Paulo, Brazil
kvd@usp.br

***Abstract*—This work proposes a new approach for the graph-based procedural level generation for games, through evolutionary algorithms. The levels are encoded as a graph structure inspired by the concepts of a scale-free network. A scale-free network lacks an internal scale as a consequence of the coexistence of nodes with different degrees in the same network. This approach aims to avoid the generation of linear, repetitive and grid-like levels, giving the algorithm additional freedom to explore the search space for diverse solutions. The algorithm was designed to provide a smooth mixed-initiative authorial control, allowing the designer to adjust constraints parameters, input aesthetically desired properties, manage critical contents of the level and even edit edges and nodes in a drag-and-drop manner. The results show that the algorithm can evolve scale-free structures with moderated nonlinearity, which is taken as an ideal measure for game levels that feature player progression like lock-and-key puzzles.**

***Index Terms*—procedural content generation, scale-free networks, evolutionary computation**

I. Introduction

In the field of Procedural Content Generation (PCG) for computer games, one of the most common generation problems is the automated creation of levels [1]. Here, a *level* is defined as the spatial environment where the players can walk through during the gameplay. Despite being a common problem, the task of generating a reasonable level is still hard and it is usually handled by search-based methods, such as Evolutionary Algorithms (EA) [2].

One of the benefits of using PCG for the creation of computer games is the expected reduction of the costs in the team's budget [3]. There are also other benefits like the diversity of the solutions, the rapid content generation, and others. However, despite such good benefits, complete search-based level generators are still unpopular in the industrial field [4], in comparison with simple single content generators such as SpeedTree [5]. A possible reason for this unpopularity is the lack of control that human designers can exert on the creative process and the unpredictable nature of the stochastic metaheuristics used.

Therefore, a desirable property of PCG techniques is called *controllability*, defined as the ability to control the generation process, setting desired aesthetical features, evaluation metrics, among other essential characteristics [1]. In such scenario, the system can run entirely independent or aided by a human designer in a mixed-initiative approach [6].

The goal of this paper is to provide a mixed-initiative tool for generating game levels, providing an improvement to the state of the art solutions. To demonstrate the method, we defined the level as a three-dimensional *dungeon*, a typical environment to adventure games or Role Playing Games (RPG) styles. In this dungeon, the player must reach a goal, moving through the rooms and eventually solving some puzzles. These puzzles can be in a lock-and-key style, where the player needs to find a key to proceed.

There are a few successful examples in literature, such as the already mentioned works [3], [4]. Though, these works usually generate maps with predictable degree distribution among the nodes, consequently resulting in solutions with reduced diversity in the overall dungeon topology.

This paper presents a novel approach, representing the content as scale-free complex-network [7], instead of using graph grammar-based solutions [8] or other grid-like and tile-based mechanisms [9]. Our results show that the proposed method can evolve diverse, resolvable, compact and nontrivial solutions with user-controlled nonlinearity and desired aesthetical characteristics. The developed code is available at our GitHub repository[1].

II. Related Work

As already mentioned, there are many successful methods for the procedural level generation problem, through a wide range of different techniques, which are not confined to just EA [1]. For example, in [10] the authors used a Cellular Automata model to generate caves. Despite its efficiency, it is tough for the designers to control the generation process and define the attributes of the output content.

Recently, some papers have used graph grammars to generate levels for adventure games [3], [4], [11]. In such models, the graph structure represents first a mission (or a quest) plot that motivates the spacial level generation. Thus, driven by missions, the algorithm can generate meaningful levels.

Other similar works [12], [13], also consider a story plan at first place, using tree structure to represent levels, which is also the genetic representation. Probably, the most similar work is [14], where the authors used a graph to describe a 3D dungeon and the contents were generated considering a set of constraints.

[1] https://github.com/KuruLab/ScaleFreeMapGenerator

978-1-5386-4359-4/18/$31.00 ©2018 IEEE

Different from the papers mentioned above, this work does not focus on the mission, quest, or story plan as a motivation for the level generation. Even in featuring lock-and-key puzzles, the puzzles do not motivate the algorithm's creative process. The lock-and-key puzzles are taken as a secondary step in the individual decoding. However, we do not discard the possibility of implementing this successful idea in future works.

On the other hand, all the mentioned works, including [14], share a common limitation: the level topology consists of a grid or follows predictable rules. Especially in [13], there is an explicit grid mapping stage. Consequently, it is possible to figure out not just a pattern in the degree distribution of the rooms (as nodes of a graph), but also the angular orientation of the room's connections. In this paper, we focus on the generation of the spatial level itself, proposing a novel individual representation, inspired by scale-free networks [7], which does a step forward to overcome such limitations.

In this paper, we consider the lack of topological diversity as a drawback that we attempt to overcome. For example, in [3], the dungeons are generated for Zelda-like games [15], but the grid-like shape of the resulting levels converts it into an *ad-hoc* application. It works for Zelda-like rooms, but it may fail in producing 3D levels for realistic adventure games, where not all the rooms are arranged orthogonally.

The graph grammar-based level generators evolve levels where the number of connections between nodes happens predictably. In such cases, there is only one connection in each cardinal direction, and it is barely possible to see levels featuring simple diagonal connections. Additionally, fixed parameters set the number of nodes or the level dimensions. This work can overcome these limitations, showing solutions featuring free-form topology, which in turn can be controlled by a human designer, with granted resolvability and nontriviality.

III. Background

A scale-free network is a type of complex network that has the connections (edges) between its nodes determined by a stochastic algorithm. The nodes in this network are inserted one by one, and whenever a new node is inserted, it makes m, $m \geq 1$, connections with other nodes already existing in the network. The node to which the new node will connect is determined probabilistically. The probability to connect to a node is proportional to the number of connections it already has (degree). As the number of nodes in the network grows, the trend is that a few nodes will have the most significant number of connections, usually called *hubs*.

The inspiration in this model is justified by its capabilities to describe networks that grow infinitely (in theory). Additionally, it is not possible to predict the average degree of the network as it grows, hence the name scale-free. Accordingly to Barabási, "the scale-free name captures the lack of an internal scale, a consequence of the fact that nodes with widely different degrees coexist in the same network" [7].

The traditional Barabási-Albert model constructs the network in a stochastic way, but in [7], chapter 5, an alternative deterministic model is presented, oriented towards optimization in real-world applications, taking into account Euclidean distances between the nodes. For this reason, in this work, we adopted the mentioned network model. In this alternative model, hubs feature a *basin of attraction* that lures nearby nodes to link (i.e., create an edge) to it [16]. An iterative process builds the network. On each step, a new node is inserted in the network at random positions and attach itself to already existing nodes. The parameter m, $(m \geq 1)$, defines the number of new links. The m new links from the new node i are created based on cost function [16]:

$$C_i = \mathtt{min}_j[\delta d_{ij} + h_j], \tag{1}$$

where d_{ij} is the Euclidean distance between node i and a candidate node j, already inserted, h_j is the network-based distance (path cost) from node j to the very first inserted node, considered as an ideal network center. Thus, the new node i will link itself to the least cost node j. The δ factor is related to the strength of the basin of attraction. For example, if $\delta = 0$, the Euclidean distances are irrelevant. Hence each node links to the first central node, shaping the network as a star. It happens because of $h_0 = 0$, and so the central node is always the best one. Star topologies can be observed for $\delta \leq (1/2)^{1/2}$. However, for large values of δ, each node is connected to the nearest neighbor, because the contribution provided by the distance term δd_{ij}, in (1), dominates h_j. In this case, the network will have a random topology. On the other hand, Barabási guarantees that the deterministic model can present the same characteristics of a scale-free network for $4 \leq \delta \leq N^{1/2}$, where N is the total number of nodes. Therefore, the oblique boundary of the scale-free regime is $\delta \leq N^{1/2}$.

The power law distribution in this model has its origin tied to both optimization and randomness, even without a stated probability function. It is known that these mechanisms lead to linear *preferential attachment*, as assumed in the referred Barabási-Albert model. The read of [7] is strongly suggested for a better understanding of such complex models.

IV. Proposed Approach

In the this section, we describe our proposed approach, detailing the adopted network model and the implemented Evolutionary Algorithm (EA).

A. Network Model

In this work, our adopted network model differs briefly from the method described in Section III. Basically, we did two minor changes:

- First, instead of disposing a node at a random position, the node positioning is defined by evolutionary means.
- Second, each new link is created with a probability $p_\ell = 1/\ell$, where ℓ is the link number, such that $1 \leq \ell \leq m$.

For example, for $m = 2$, the first link ($\ell = 1$) is always guaranteed to exist, since $p_1 = 1$, but the second link ($\ell = 2$) has a $p_2 = 0.5$, thus having half chances of not existing. Since $m = 1$ results in an acyclic graph, the default value for this

parameter is $m = 2$. As $\delta \leq N^{1/2}$, and because we deal with networks as small as 25 nodes, the δ value was fixed in $\delta = 5$ in this whole work.

In this abstraction of the level as a complex network, the nodes of the network correspond to physical regions in which the player can walk, and the edges correspond to the connections between those regions. It is possible to imagine different ways of mapping the network graph in a 3D game level. For example, a single door can represent a connection between two rooms, as well as a bridge (between two sides of a river), a road (between cities), and so on. In this work, we consider an adventure game, where the level is a dungeon, consisting of rooms (nodes) and corridors (edges) at their most basic abstraction. However, it is easy to extend this work for other level abstractions and game genres, with minor changes in the proposed generator code, since its output is just a network description.

All network data is stored in JSON formatted files, allowing easy human reading and writing, as well as compatibility with several other systems and languages. The Unity3D[2] game engine does the network mapping into a virtual representation as a 3D game level, using a deterministic algorithm, implemented in C# language. Thus, C# scripts interpret the data from a JSON file and create the 3D environment with the characteristics described above: nodes are parsed into rooms and edges are parsed into paths connecting the rooms. When the game scene is loaded, the player is automatically positioned at the center of the start room. The goal of the level is to reach the boss. However, there are many locked rooms and the path to the boss may not be trivial. To overcome this problem, the player should find keys along his path to unlock the rooms and find a way to the boss. This simple game description imposes a few inherent rules that are not present in the scale-free network model. These rules are handled as problem constraints, which we discuss below.

B. Emergent Constraints

Complex-networks often show many intriguing structures of edges sharing the same source node. In the abstract space of a network, represented by a graph, a vertex does not have physics or doesn't occupy space. The same applies for an edge. Thus, in a visual representation of a graph, if an edge crosses another edge, it does not mean the edges are connected, cause edges do not link with other edges, but only vertexes. Even if an edge crosses another vertex, it is possible that this vertex is neither source (origin) nor the target of the edge. Therefore, in the geographical (virtual) representation of a network, if there is no treatment to such conditions, it is possible to have corridors blocked by room walls (without doors, because they are not connected), which could break the level, making it unsolvable. Additionally, edges crossing might be translated into unpredicted corridors crossing, creating shortcuts, which in turn can trivialize the level. It is also possible to imagine other undesired situations.

[2]Unity3D: https://unity3d.com

To avoid the problems described, there are three constraints related to nodes and edges geography. First, nodes cannot have any overlapping internal area, although they can share a wall. Second, edges cannot intersect nodes or other edges and for this purpose, not only the straight segment is considered, but the area formed by the width per length of the corresponding corridor. Third, if two nodes are connected diagonally, the edge's length must be higher than a minimum value (currently defined as 5 meters). The third constraint was designed to prevent the installation of doors in improper positions, when the connected rooms are sharing a corner. However, it can be removed in future versions if a better algorithm for parsing edges into corridors is conceived. Currently, the corridors are straight paths, without corners or curves.

There is a fourth constraint: the number of nodes must be higher than a predefined minimum. It was created to avoid the generation of tiny dungeons and to prevent the network from running out of the scale-free regime (remember $4 \leq \delta \leq N^{1/2}$), ending up in a random or a star shape. We also define a maximum number of nodes, to prevent a network overgrowth, despite the fact that it is doubtful to happen.

Since scale-free networks inspire this design, there is no restriction to the number of edges or degree of the nodes. However, in practice, the constraints mentioned above end up forcing the algorithm to avoid high degree nodes, since the excess of adjacent rooms may eventually violate some constraint (especially if we consider that the corridors have a fixed width). Similarly, the length of a corridor is unlimited, but it is worth remembering that the more extended its length, the higher the chances of intersecting with other corridors or rooms.

C. Content Representation

The system's goal is to produce a game level for an adventure game. The level is, in turn, built by parsing a complex-network into a 3D virtual environment. However, the individual *phenotype* is the network structure and not the game level itself. Following, we describe the *genotype* encoding and the decoding process.

There are many successful proposals of how to represent a network as an individual. For example, [17] presented the Node-Depth Encoding (NDE) for network design problems. Despite the reported NDE efficiency for building minimal spanning trees, it is suitable for producing acyclic graphs.

In this work, the acyclic graph generation is considered as a drawback, since it results in a level featuring a linear player progression, a feature that we attempt to avoid. Thus, cyclic graph representation is desirable, due to its potential to produce more alternative paths to the player's goal. Additionally, the generation of edges by evolutionary means and not by the methods described in Section IV-A can compromise the scale-free network characteristics, unless we design the genetic algorithm explicitly for this task (which is not the case). For this reason, we decide to omit the edges in the encoding of an individual. In other words, the genotype of an individual consists of only an array of nodes.

Under these circumstances, to build up an entirely connected graph that represents a dungeon, the decoding process has two sequential steps:

- The *connection step*, to link the nodes.
- The *dungeon step*, to create a lock-and-key puzzle.

The connection step follows the method described in Section IV-A. Consequently, the evolutionary process has the task of finding suitable positions for the rooms of the dungeon. However, the scale-free regime takes not only the nodes' coordinates but also their order. In the *i*-th iteration, the *i*-th node is inserted. The method considers the first node as the network center and the most critical node, since each subsequent node will take into account the cost to connect to it. Also, while forging the new links, it computes the cost function (1) only for the previous nodes, i.e., for the nodes that are already present in the system. Thus, the order in which the nodes are arranged in the genetic material matters. Hence, the EA is responsible for not only optimizing the nodes positioning at the virtual space but also the nodes ordering at the encoding array.

In [17], the authors also point that the generated graphs may be very different from their parents, depending on the adopted representation. As a consequence of this inconvenience, the algorithm may present a very slow convergence. In fact, it is expected that, in a suitable representation, small changes in the genotype results in proportionally small changes in the phenotype and the fitness value, thus, preserving the principle of locality. According to Section IV-A, our decoding process establishes the number ℓ of links of a new node randomly. Therefore, a child that receives a node may generate a different number of links for that node. In other words, the same node set may show a different edge set within different individuals, which is considered to be a problem.

To solve that problem and preserve the desired properties, we control the randomness of the connection step. The pseudo-random number generator takes a seed as input. This seed is the hash code computed over the node coordinate set. By this way, two nodes at the same (x, y, z) point (even if in different individuals) produce the same hash code. Consequently, as the seed is the same, both nodes share an equal number of links. Hence, the genotype-phenotype mapping problem is solved.

When the connection step is complete (i.e., all nodes are inserted and all edges are computed), there dungeon step starts. The network is submitted to an extended version of the MetaZelda[3] project. The MetaZelda project is a Java program used to create Zelda-like dungeon scratches. It iteratively creates nodes in a tree structure to ensure a connected dungeon of predefined size. Then, it generates a lock-and-key puzzle, while ensuring resolvability and finally converts the tree into a graph. In our extended version, the graph was already created by the previous connection step. Instead of building a new random tree, it generates a spanning tree from the given connected graph. Based on this tree, it then creates the lock-and-key puzzle, defining the start room, the goal room, the keys and doors disposition. Finally, the edges outside the spanning tree are fixed to prevent the dungeon trivialization (i.e., it places proper doors if necessary).

The idea of using an extended version of MetaZelda was just to test the nonlinearity capabilities of the initial results. However, doing this process for every individual in the population, in every decoding, is not an efficient practice, since the population may contain unfeasible solutions that cannot even hold a puzzle. Despite being very useful at first moments, the MetaZelda mechanisms are not based on search algorithms. Thus, in future versions, we consider delegating this task to another search-based process. For example, in [9], the authors used cooperative coevolution to evolve vital elements for a game level, in which each subpopulation was designed to evolve just a particular kind of content.

D. Evaluation

Although some restrictions have been presented in IV-B, the evaluation function has not been defined yet. For each constraint, there is a penalization method that measures how much the individual violates the restriction. That is, the algorithm allows the existence of infeasible solutions, but penalizes them, so that their probability of survival is low. At its basic level, the algorithm's goal is to minimize the penalization value. For this task, the results of the four penalization methods are summed up without any distinction between them. However, there are other objectives, called *design preferences* that we discuss ahead.

In [18] and [3], the authors defended the idea of building a user-friendly system, which allows the designer to control the content generation. In this work, we applied the same principles to implement a collaborative mixed-initiative design process. There is a graphical user interface in which a human designer can add objectives and tweak several parameters at runtime. The user interface is built over the GraphStream[4] framework. This framework provides graph viewers where the human designer can manipulate vertexes by clicking and dragging with a mouse. Additionally, in the user interface, the designer can toggle on or off the presence of design preferences that affect the individual evaluation.

There are currently four design preferences. They were initially conceived to work independently. Nevertheless, they can provide exciting results while working together. Instead of using a Multi-Objective approach [19], it was decided to keep with a single objective algorithm that first prioritizes the minimization of the penalizations and then takes the design preferences into account as a tiebreaker (or a second fitness function). The idea was to keep the algorithm simple enough at early stages of development and evaluate the effect of the design preferences over the evolutionary process. However, we do not discard the possibility of extending each design preference as a new objective, driving the problem to the many-objective optimization field [19].

The first design preference is to maximize the number of existing nodes in the network. Thus, if two individuals with

[3]MetaZelda GitHub: https://github.com/tcoxon/metazelda

[4]GraphStream: http://graphstream-project.org/

zero penalization (or equal) are competing for survival, the tiebreaker consists in prioritize the individual with more nodes (N). The idea of this measure is not to say the more rooms, the better, but to investigate how much the individual can grow within our network model (see IV-A), without violating any of the mentioned constraints (see IV-B). It is challenging to define the ideal number of rooms in a dungeon or set the bounds of minimum and maximum values. That relies on personal feeling and can vary drastically from game to game. While in [18] a procedurally generated dungeon presents 27 rooms, in [3] the authors takes as reference the Gnarled Root Dungeon, from The Legend Of Zelda: Oracle of Seasons [15], with only 20 rooms. In this work, we set the minimum as 25 and maximum at 100, and it does not require formalization.

It is important to highlight that the network has no bounds for its geographical space. Thus, the second design preference is the minimization of the all-pairs *Average Shortest Path length* (ASP) , in an attempt to control the network geography, preventing the existence of long edges. We formalize the ASP as follows [7]:

$$\text{ASP} = (N(N-1))^{-1} \sum d_{ij}, \tag{2}$$

where N is the total number of nodes, and d_{ij} is the travel distance of the shortest path from node i to node j, $i \neq j$, which is in turn computed by the GraphStream's A* algorithm. We chose the ASP metric based on the fact that it carries more information about the network than just the edge's length. By trying to minimize the ASP, the algorithm is indirectly creating convenient edges between scattered nodes. Those additional edges may even help the generated dungeon to feature a higher number of alternative paths. Consequently, the adoption of this metric can circumvent the generation of linear levels.

One of the desired properties of a procedurally generated content is its *creativity* and *believability*, or the ability to seems like a computer did not create it, but a creative human artist [1]. Human-designed artifacts usually present patterns, symmetry, and familiar geometrical shapes. The third design preference relies on this aesthetical field. It is called *Undesired Angular-difference Sum* (UAS), and as its name suggest, the algorithm attempts to minimize it. The user specifies an array $A = \{a_0, a_1, \ldots, a_j\}$ of desired angles (in degrees). For each edge i, the algorithm computes the absolute difference between the edge's rotation θ_i, $0° \leq \theta_i \leq 180°$, and each element in A and, then, picks the minimal value. This UAS value is the sum of all minimal results. If θ_i matches with a specified angle a_j (i.e., $\theta_i = a_j$), the minimal difference is zero, and so it takes no effect on the UAS. We formalize the UAS as follows:

$$\text{UAS} = \sum_{i}^{E} \texttt{min}_j[|\theta_i - a_j|], \tag{3}$$

where E is the total number of edges and $a_j \in A$. While trying to minimize the UAS of the individuals, the algorithm produces solutions that fit the designer aesthetical preferences. It is important to remind that one of the primary goals of this approach is to evolve solutions that feature a free-form, avoiding the grid-like shaped answers present in literature [18]. However, the designer controllability is also a primary goal, and this method gives him the power to influence the evolution of the dungeon geometry. Furthermore, to keep the free-form pattern, the designer can just turn off the UAS minimization.

Finally, the last design preference is the *Distance from the Ideal Nonlinearity* (DIN) value, in which the algorithm also attempt to minimize. In this context, a nonlinear level is a level that features path backtracking to solve it. In the MetaZelda project, the nonlinearity score, $\text{NL}_{\text{actual}}$, is defined as the number of times a player must walk through each room after its first encounter, on the shortest path through the dungeon. In [20], Mike Sout discuss the concept of nonlinearity present in Zelda dungeons. Despite considering a linear level design as a bad feature that we aim to avoid, although, we agree with the author's general idea: high values of nonlinearity are undesirable. For instance, the Gnarled Root Dungeon, discussed in [3], has a nonlinearity score of 3 ($\text{NL}_{\text{actual}} = 3$). For this reason, the default value of the ideal nonlinearity NL_{ideal} parameter is set to 3. We formalize the DIN value as follows:

$$\text{DIN} = |\text{NL}_{\text{ideal}} - \text{NL}_{\text{actual}}|. \tag{4}$$

It is important to remember that each of the four mentioned design preferences could be turned on or off at the runtime. By this way, it is possible for the designer to change the fitness function at the 50-th generation, for example, forcing the current population to adapt to the new environment.

E. Genetic Operators

Due to the chosen individual representation, the rest of the genetic operators are easy to conceive and understand. The individuals undergo selection employing a Binary Tournament, in which two individuals are compared, selecting for mating the individual with the best fitness.

The crossover has probability $\rho_c = 0.9$ and occurs with a random cut-off point, producing two children consisting of the permutation of the elements. On important detail is that the number of nodes in each individual is undefined. For this reason, it is necessary to treat this cut-off point so that it does not exceed the size of one of the individuals. It is important to mention that the simple cut-off point can cause identical nodes, commonly shared between parents, but at different positions of the array, to be both passed to a single child, creating redundancy. For the evolutionary algorithm, this is not a problem, because the first constraint (the restriction of intersection area between nodes) penalizes this solution.

The mutation of each child generated occurs with a global probability $\rho_m = 0.1$. There are four mutation operators, all with the same $\rho_m/4$ probability: (i) adds a new randomly generated node; (ii) removes a randomly selected node; (iii) exchange the position of two randomly chosen nodes; (iv) disturbs the coordinates (x, y, z) of a randomly selected node.

It is worth mentioning that the coordinate system starts at the point $(0, 0, 0)$, and there is no negative coordinate. There is a b_{size} border value that applies to the x and

y axes. Therefore, the minimum coordinate for a node is $(b_{size}, b_{size}, 0)$. However, there is no maximum coordinate, and the individuals are free to expand its dimensions as much as possible. The width and the height of an individual are defined by the farthest node in the respective axis.

Each coordinate has a parameter that controls the possible tweaking values during the mutation operators. These are the parameters v_x, v_y and v_z. They refer to the maximum perturbation that a coordinate can suffer, both positively and negatively, in said axis, concerning the global dimensions of the level. The limits of the perturbation can be computed as $\texttt{bounds}_x = x \pm ((v_x - 1) \times \texttt{width})$ for the x axis. Thus, in each mutation, a level has the potential to enlarge or reduce its dimensions, proportional to the value of these parameters.

There is also a refinement operator, which runs with probability $\rho_r = 0.01$ on each individual. The refinement consists of a *first improvement* heuristic that attempts, iteratively, to reposition the nodes in supposedly ideal positions. If the repositioning of a node results in a reduction in fitness, then the modification is accepted. Otherwise, it discards the change. The refinement ends when no development is found in the neighborhood structure around the solution in improvement.

After each improvement on the best fitness value found so far, the respective best individual is submitted to the refinement operator with $\rho_r = 1.0$, and then it is stored in an archive. At the end of the reproduction steps, each child overtakes its parents' position in the population. There is no elitism. The algorithm has a population of $\mu = 100$ individuals and runs for a maximum of $g = 100$ generations.

V. Computational Experiments

In this Section, we present the computational results of the conducted experiments. It is important to remind that, even producing a 3D level, the algorithm is not working over the z axis yet, and so $v_z = 0$. It is also important to remember that there is no restriction on the maximum length of an edge, but there is the minimum value, $d_{min} = 5$, which only applies in a particular case (corner encounter), described in Section IV-B. The size of a room, in the current version, is normalized automatically, based on the node degree, taking into account the parameters $S_{min} = 15$ and $S_{max} = 30$.

Since there are four metrics to evaluate a solution, in this work different scenarios were considered, in which the configuration of the evaluation function is varied. In the standard scenario, all four metrics work together. In this case, the desired angle set is $A = \{0°, 90°, 180°\}$ (orthogonal connections). Despite trying to avoid grid-like levels, in general, here we want to prove that the algorithm can find solutions featuring connections approximately at any desired angle set, including grid-like shape if the designer wants to. The Table I presents the results of a total of 30 runs of the standard experiment. It distinguishes the minimum, mean (and standard deviation) and maximum values, for each metric, independently. The N value is the number of nodes. The area value (which is in the 10^5 scale) is not an optimizing metric, but it is shown here to promote future comparison. The algorithm successfully reaches a 100% accuracy regarding the DIN metric, i.e., it always returned solutions with nonlinearity score of exactly 3. The first impression is that the algorithm is free to produce levels within the size bounds. However, that cannot be observed. Even rewarding solutions with higher N, the algorithm ends up minimizing N, reaching the lower bound $N_{min} = 25$ (or approaching, with 26 in just one case) in all executions. This fact demonstrates the conflicting nature of the valued metrics. To understand that, consider the ASP minimization problem. The algorithm may create new edges, which serves as shortcuts, in attempt to reduce the ASP value. However, if the shortcut edge's rotation does not match with a desired angle in the A set, its creation results in a consequent increase in the UAS value, thus increasing the fitness value. By this way, reducing the ASP by creating new edges is not that trivial task, since it may imply a worse fitness. Then, it is necessary to find suitable positions in a matching rotation to achieve a good trade-off between such metrics, of course, taking care of every constraint. Alternatively, the algorithm can just remove a node. There is no edge to a non-existing node. Thus, there is no path, no rotation angle, no intersections, and so on. Consequently, removing a poorly positioned node is a more straightforward solution to achieve a higher fitness value. Conclusively, by trying to minimize both ASP and UAS, the algorithm also minimizes N, until the lower bound.

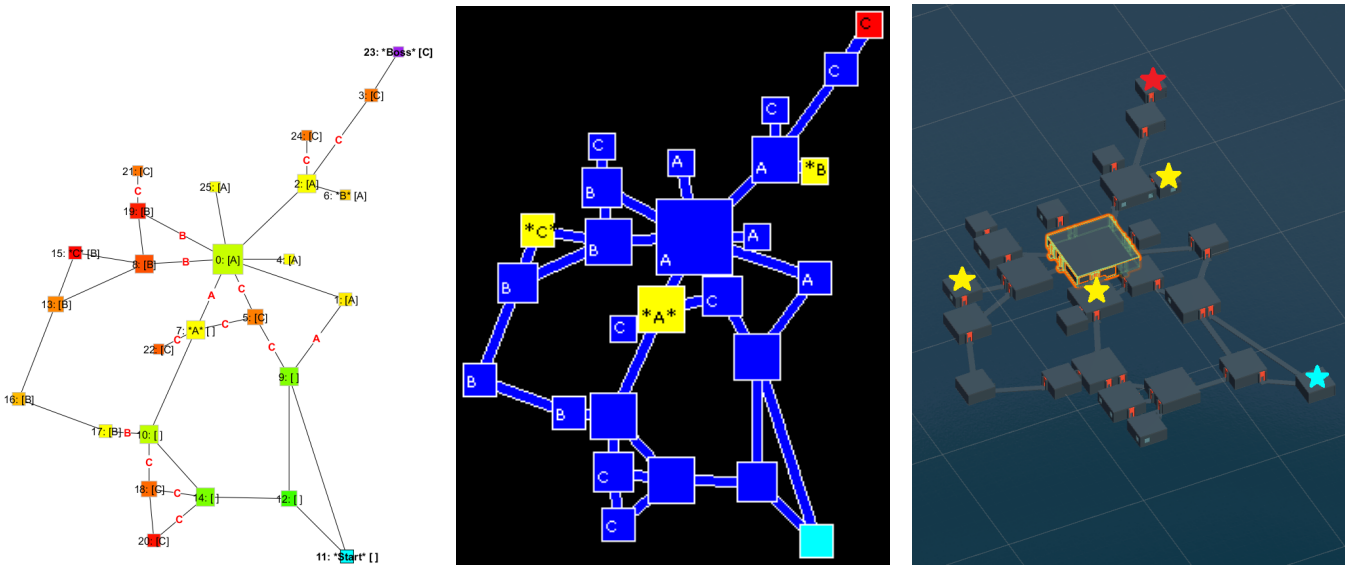
TABLE I
Standard Scenario Results

Metric	Min	Mean ± Std	Max
N	25	$25.033_{\pm 0.183}$	26
ASP	260.836	$320.888_{\pm 41.642}$	419.523
UAS	28.093	$80.036_{\pm 33.547}$	149.619
DIN	0.000	$0.000_{\pm 0.000}$	0.000
Fitness	318.020	$400.924_{\pm 52.587}$	544.822
Area ($\times 10^5$)	1.6	$2.9_{\pm 0.8}$	4.8

These results suggest that, due to the conflicting behavior of the considered metrics, they may not be the most adequate to meet the initial intentions for which they were designed. The ASP design intentions were not just a compact level, but a suitable set of edges, providing alternative paths. Thus, in future versions, it may be considered a Pareto front, where each metric correspond to an additional objective, instead of a single objective function with tiebreakers.

In the second scenario, the fitness function consists of only ASP and DIN metrics thus, removing the UAS and the higher N priority. The idea of this experiment is to investigate the effect of the ASP metric without the influence of the desired angles. The Table II present the results of 30 runs of such experiment.

Comparing Tables I and II, it is easy to see that the fitness value has narrowed beyond what should be only by removing the UAS value. That relies on the problem complexity. By adding a new metric, the problem complexity increases, hence, making it harder for the algorithm to find quality solutions, as suggested by [3].



(a) GraphStream's Viewer (b) True Map (c) Unity3D's Scene Editor

Fig. 1. Best individual from the ASP and DIN scenario.

TABLE II
ASP AND DIN SCENARIO RESULTS

Metric	Min	Mean ± Std	Max
N	25	$25.167_{\pm 0.658}$	30
ASP	138.428	$168.829_{\pm 14.542}$	206.222
DIN	0.000	$0.078_{\pm 0.269}$	1.000
Fitness	138.428	$168.906_{\pm 14.498}$	206.222
Area ($\times 10^5$)	0.821	$1.283_{\pm 0.211}$	1.896

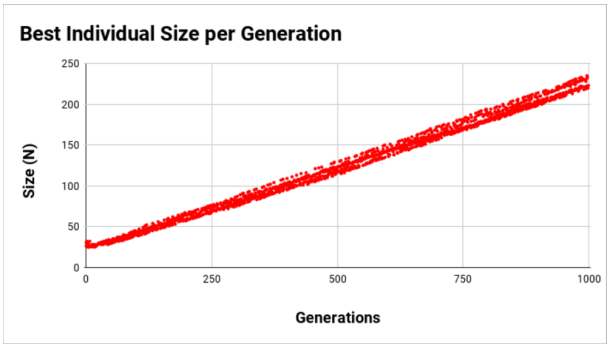


Fig. 2. Third Scenario: the size of the best invididual over the generations

To illustrate a result of the current work, Fig.1 shows three different versions of the best individual in the second scenario. Figure 1a shows GraphStream's version. In this figure, the teal node, at the bottom right corner, is the starting point, while the purple node, at the top right, is the boss (goal) room. For each other node, the colors are a heat map of the relative difficulties of the rooms in the dungeon, in such a way that: green means easy; yellow means medium; red means hard. However, despite computed, such difficulty is not yet taken into consideration in this work, consisting of a feature out of the scope of this paper. A few edges present a letter on it, representing the required key the player must have to traverse it. The same applies to the nodes' letters inside the brackets. Figure 1b consists of a true map representation, where the actual dimensions of rooms and corridors are translated. The teal room is the starting point, the red is the goal, the yellow ones are the keys placing, and the blue ones are just normal rooms. The normal letters mark the preconditions (i.e., the key the player must have) to enter the respective room, while letters between stars mark the actual key. Finally, Fig.1c shows the same individual at runtime on the Unity3D's scene editor. The small red points are the doors, while the highlighted central room is the network center (i.e., the first node).

Finally, the third scenario was designed to investigate the solutions' growth capabilities. In this scenario, the ASP, UAS and DIN metrics are toggled off. Thus, the fitness function consists only in, first, minimize the violation penalties, including the size bounds, and second, maximize N. There is no limit for the level's dimensions, which is directly influenced by the v_x and v_y parameters. However, the insertion of a new node has spatial implications, such that the higher the N, the harder it became to add a new node without violating any constraint. Thus, the complexity of the algorithm grows substantially as N increases over the generations. Therefore, the algorithm requires more generations to reach considerable high values of N.

In earlier experiments, we believed that such spatial issues could lead to a "practical bound" to the growth of individuals and other scale problems. However, we were wrong about that. In fact, for $g = 100$, the maximum value of N found was 48. We decided to test different settings of v_x and v_y parameters in several experiments, resulting in large and sparse maps.

The results demonstrated that, even in a considerably large (although undesired) area, the supposed practical bound of $N = 48$ was still very consistent, thus, revealing its nature does not rely on a dimensional problem, but on other evolutionary parameters, such as the stopping criteria ($g = 100$). For this reason, we decided to let the algorithm run for $g = 1000$ generations. With this extended stopping criteria, it is possible to find solutions with up to 235 nodes.

The Fig.2 presents a scatter plot of the size of best individuals found at its respective generation in another 30 runs. As can be seen, the growth of N over the generations follows a linear trend within the observed range. Thus, this result made us to discard the idea that there is a potential problem between the allocation of nodes and the spatial growth of the map.

It is important to mention that in all above experiments, totalizing 90 executions, not a single violated solution was returned, showing the strength of this model in finding quality solutions. In general, we can affirm that the algorithm can overtake the literature appointed limitations, however, presenting its drawbacks regarding the higher complexity. Even without using a search-based algorithm to build the lock-and-key puzzles, the algorithm is successful in achieving the desired nonlinearity score.

VI. Conclusions and Future Work

This paper presented a novel approach for the graph-based level generation problem. It distinguishes from the current literature by using a complex network model, specifically the Barabási-Albert scale-free model, to represent the procedurally generated content, instead of tree structures, acyclic graphs or graph grammar-based models. An EA was used to evolve levels considering four penalization functions and four design preferences. We also provided a user interface, where a human designer can interact with to control the generation process.

In literature, the most prominent solutions usually are limited to create grid-like topologies for the dungeons. Often, the rooms' connections are tied to one neighbor in each cardinal direction. The goal of this paper is to provide a tool that is easy to use and can produce levels in a large variety of topologies, either automatically or aided by a human designer. The results indicate that the algorithm is successful in achieving this goal, even considering different scenarios. However, the data shows the algorithm suffers from a natural limitation: the addition of one or more metrics in the evaluation function increases the algorithm's complexity, thus affecting the quality of the final solutions negatively.

Shortly, we plan to extend the current version to a many-objective approach, where each metric corresponds to an objective. This extension will allow a better investigation of the design preferences, due to its conflicting nature. Another probable extension consists of delegating the task of creating a mission plan, and the lock-and-key puzzles, to a separated process, thus, simplifying the spatial level generator. Additionally, it could be implemented another method to populate the rooms of the dungeon, accordingly to the already computed difficulty, providing an appropriated level of challenge. All parallel processes can be integrated through a coevolutionary approach or by a multi-agent system, where each process is responsible for a specific task, but in cooperation with each other to produce a complete solution.

References

[1] N. Shaker, J. Togelius, and M. J. Nelson, *Procedural Content Generation in Games*. Springer International Publishing, 2016.

[2] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne, "Search-based procedural content generation: A taxonomy and survey," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 3, no. 3, pp. 172–186, sep 2011.

[3] J. M. Font, R. Izquierdo, D. Manrique, and J. Togelius, "Constrained level generation through grammar-based evolutionary algorithms," in *Applications of Evolutionary Computation*, G. Squillero and P. Burelli, Eds. Cham: Springer International Publishing, 2016, pp. 558–573.

[4] R. van der Linden, R. Lopes, and R. Bidarra, "Procedural generation of dungeons," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 6, no. 1, pp. 78–89, mar 2014.

[5] I. Interactive Data Visualization, "Speedtree," 2017. [Online]. Available: http://www.speedtree.com/

[6] A. Liapis, G. N. Yannakakis, and J. Togelius, "Sentient sketchbook: Computer-aided game level authoring," in *In Proceedings of ACM Conference on Foundations of Digital Games, 2013. In Print*, 2013.

[7] A.-L. Barabási, *Network Science*. Cambridge University Press, 2016. [Online]. Available: networksciencebook.com

[8] M. H. Luerssen and D. M. W. Powers, "Graph design by graph grammar evolution," in *2007 IEEE Congress on Evolutionary Computation*. IEEE, sep 2007.

[9] M. Cook, S. Colton, and J. Gow, "The ANGELINA videogame design system—part I," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 9, no. 2, pp. 192–203, jun 2017.

[10] L. Johnson, G. N. Yannakakis, and J. Togelius, "Cellular automata for real-time generation of infinite cave levels," in *Proceedings of the 2010 Workshop on Procedural Content Generation in Games - PCGames '10*. ACM Press, 2010.

[11] J. Dormans and S. Bakkes, "Generating missions and spaces for adaptable play experiences," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 3, no. 3, pp. 216–228, sep 2011.

[12] V. Valtchanov and J. A. Brown, "Evolving dungeon crawler levels with relative placement," in *Proceedings of the Fifth International C∗ Conference on Computer Science and Software Engineering - C3S2E'12*. ACM Press, 2012.

[13] K. Hartsook, A. Zook, S. Das, and M. O. Riedl, "Toward supporting stories with procedurally generated game worlds," in *2011 IEEE Conference on Computational Intelligence and Games (CIG'11)*. IEEE, aug 2011, pp. 297–304.

[14] T. Roden and I. Parberry, "From artistry to automation: a structured methodology for procedural content creation," in *Entertainment Computing – ICEC 2004*. Springer Berlin Heidelberg, 2004, pp. 151–156.

[15] Nintendo, "The legend of zelda: Oracle of seasons," 2001.

[16] A. Fabrikant, E. Koutsoupias, and C. Papadimitriou, "Heuristically optimized trade-offs: a new paradigm for power laws in the internet," in *Proceedings of the 29th International Colloquium on Automata, Languages, and Programming (ICALP)*, Malaga, Spain, Jul. 2002, pp. 110–122.

[17] A. C. B. Delbem, A. de Carvalho, C. A. Policastro, A. K. O. Pinto, K. Honda, and A. C. Garcia, "Node-depth encoding for evolutionary algorithms applied to network design," in *Genetic and Evolutionary Computation – GECCO 2004*, K. Deb, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 678–687.

[18] D. Karavolos, A. Bouwer, and R. Bidarra, "Mixed-initiative design of game levels: integrating mission and space into level generation," in *Proceedings of FDG 2015 - Tenth International Conference on the Foundations of Digital Games*, jun 2015.

[19] B. Li, J. Li, K. Tang, and X. Yao, "Many-objective evolutionary algorithms: A survey," *ACM Comput. Surv.*, vol. 48, no. 1, pp. 13:1–13:35, Sep. 2015.

[20] M. Stout, "Learning from the masters: Level design in the legend of zelda," Gamasutra, Jan. 2012. [Online]. Available: https://www.gamasutra.com/view/feature/134949/learning_from_the_masters_level_.php

Evolutionary MCTS for Multi-Action Adversarial Games

Hendrik Baier
Digital Creativity Labs
University of York
York, UK
hendrik.baier@york.ac.uk

Peter I. Cowling
Digital Creativity Labs
University of York
York, UK
peter.cowling@york.ac.uk

***Abstract*—Turn-based multi-action adversarial games are games in which each player turn consists of a sequence of atomic actions, resulting in an extremely high branching factor. Many strategy board, card, and video games fall into this category, for which the current state of the art is Online Evolutionary Planning (OEP) – an evolutionary algorithm (EA) that treats atomic actions as genes, and complete action sequences as genomes. In this paper, we introduce *Evolutionary Monte Carlo Tree Search* (EMCTS) to tackle this challenge, combining the tree search of MCTS with the sequence-based optimization of EAs. Experiments on the game *Hero Academy* show that EMCTS convincingly outperforms several baselines including OEP and an improved variant of OEP introduced in this paper, at different time settings and numbers of atomic actions per turn. EMCTS also scales better than any existing algorithm with the complexity of the problem.**

***Index Terms*—game tree search, Monte Carlo Tree Search, strategy games**

I. Introduction

Computer programs typically play adversarial games with a form of search, choosing paths to desirable future game states as determined by e.g. a heuristic evaluation function. *Monte Carlo Tree Search* (MCTS) [1], [2] is the state of the art search framework for a variety of classical board games with moderate branching factors of up to a few hundred [3], as well as many card games, video games, and non-game domains [4].

However, most *turn-based multi-action adversarial games* – games in which each turn consists of a sequence of atomic actions, instead of just a single action – have much higher branching factors. This class of games includes board games such as Arimaa and Risk, mobile games such as Battle of Polytopia, and PC games such as Civilization, XCOM, Heroes of Might and Magic, and Into the Breach. A turn in a strategy game could for example consist of moving nine units with ten available actions each, resulting in a branching factor of one billion. Vanilla MCTS cannot handle this complexity, even with the help of various techniques for reducing the effective branching factor. Finding a good action sequence for a single turn, even without considering the next turns, is a challenging search problem in such domains. That is the problem we tackle in this paper. While some of the games in this class feature indeterminism (e.g. Risk) or partial observability (e.g. Civilization), our initial focus here is on deterministic multi-action adversarial games with perfect information.

One possible approach is searching a tree in which each edge represents an atomic action instead of a complete turn, resulting in a much smaller branching factor, but also a much deeper tree (see [5] for a similar trade-off). According to Kozelek [6] and Justesen et al. [7] however, vanilla MCTS is often not able to search the tree of its current turn deeply enough, and focuses too much on optimizing the first actions compared to the last actions. MCTS can be enhanced with pruning techniques that make the search spend the same amount of time on each action [8] – but this still suffers from the problem that MCTS has to find the actions of its turn in a fixed order, so that choices on earlier actions can influence later actions but not vice versa. Justesen et al. therefore proposed a different, tree-less search approach: *Online Evolutionary Planning* (OEP), an evolutionary algorithm that treats atomic actions as genes and complete turns as genomes [9], [7]. By searching over the space of possible next turns with the help of crossover and mutation, it can optimize each action equally and simultaneously. OEP is the current state of the art in multi-action adversarial games.

In this paper, we propose an alternative approach called *Evolutionary MCTS* (EMCTS), combining some of the features of MCTS and evolutionary algorithms. It searches a tree with nodes representing genomes (in multi-action adversarial games: complete turns instead of partial turns, or the states resulting from them), and with edges representing mutations of those genomes (in multi-action adversarial games: mutations of turns instead of additional atomic actions). EMCTS therefore explores the mutation landscape of evolutionary algorithms in a systematic, best-first manner, providing evolution with lookahead search.

We use the same testbed game as Justesen et al. [7] in this paper: the turn-based multi-action adversarial game *Hero Academy*. We also introduce an improved variant of OEP called greedy OEP by transferring some ideas from EMCTS to OEP. EMCTS is then compared to vanilla OEP, greedy OEP, and four other baseline search algorithms including two vanilla MCTS variants specifically designed for Hero Academy, at different CPU time per turn and at different numbers of actions per turn.

This paper begins with a brief review of relevant related work in Section II. Section III describes our testbed, Hero Academy, outlines the baseline algorithms we are comparing,

978-1-5386-4359-4/18/$31.00 ©2018 IEEE

and introduces Evolutionary MCTS. Section IV presents our experimental setup and results, and Section V gives our conclusions and suggests future work.

II. Background and Related Work

This section reviews work on MCTS for very large branching factors, on the current state of the art for multi-action adversarial games – Online Evolutionary Planning – and on previous attempts at combining evolution and tree search.

A. Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) [1], [2] is a best-first tree search algorithm based on stochastic simulations for state evaluation, which has been successfully applied to a large variety of games and other tasks [4]. The algorithm typically constructs a search tree with nodes representing game states, and edges representing actions leading from one state to another. In a deterministic game and ignoring transpositions, this can also be seen as a tree in which nodes represent the list of actions that have been applied from the root state to reach their respective state – this view will be helpful later. MCTS begins its search at a root node corresponding to the current game state. It then repeats the following four-phase loop until computation time runs out:

1. In the selection phase, a *selection policy* is used to traverse the tree until an unexpanded action is chosen. The selection policy should balance the exploitation of states with high value estimates and the exploration of states with uncertain value estimates. In this paper, the popular UCB policy is used [10].

2. In the expansion phase, the previously unexpanded action and a node representing its successor state are added to the tree.

3. In the rollout phase, a *rollout policy* is used to play out the remaining part of the simulated game, starting from the state represented by the newly added node. This rollout policy can be uniformly random, but can also profit from heuristic game knowledge. In this paper, we use ϵ-greedy rollouts, which select a random action with probability ϵ, and otherwise follow simple greedy heuristics.

4. In the backpropagation phase finally, the value estimates of all states traversed during the simulation are updated with the result of the finished game.

Several MCTS variants and enhancements have been proposed over time in order to apply MCTS to games with increasingly higher branching factors.

First-play urgency [11] encourages exploitation by providing a value for unvisited child nodes, removing the need for MCTS to visit every child of a node before a selection policy like UCB can be applied. Progressive widening [12] and unpruning [13] approach the branching factor problem in Go by first limiting the number of actions expanded in a new MCTS node, then growing it over time so as to improve value estimates and still guarantee convergence in the limit. For games with much higher branching factors such as real-time strategy (RTS) games, script-based approaches have been developed in order to search over a small number of hand-coded scripts instead of a larger number of atomic actions: Hierarchical Portfolio Search [14] and Script-based UCT [15] fall into this category, as well as the non-MCTS approach of Portfolio Greedy Search [16]. Some previous works have applied MCTS variants to domains with very large or continuous action spaces by making strongly simplifying assumptions such as independence of units in an RTS game [17], or similarity of "close" actions in a physics-based domain [18]. Often, the assumption is made that each unit can perform one action per time step, as is typical for RTS games. In this paper, we do not assume independence of units, do not tie actions to units, and do not assume the existence of predefined policies or scripts. We do however use a heuristic evaluation function – which is hand-coded in our test domain, but could in future work be automatically learned [3].

We are using two specifically adapted variants of MCTS as baselines in our experiments, described in Subsection III-B. The proposed EMCTS is similar to vanilla MCTS in the sense that it uses the same tree search structure of selection, expansion, rollout, and backpropagation, while working on a new, evolution-inspired search space.

B. Online Evolutionary Planning

Evolutionary algorithms (EAs) are a class of optimization algorithms inspired by natural selection that has been used extensively for evolving and training AI agents for games [19], [20]. In the classic, *offline* evolutionary approach, an AI's parameters are evolved using its performance at playing the game as a fitness function. No evolution is applied after the training has finished and the AI is deployed in the game [21], [22], [23], [24].

Online evolution is a newer approach, in which evolutionary algorithms are applied during gameplay. This can take the form of evolving the AI's parameters while it is playing [25]. However, it is also possible to evolve the next *action(s)* to take in the currently running game. *Rolling Horizon Evolutionary Algorithm* (RHEA) [26], [27] for example evolves fixed-length future sequences of actions in a single-player game, which are compared by simulating them and evaluating the resulting game states. When a time limit is reached, the algorithm executes the first action in the best action sequence found, and continues search on action sequences starting from the next time step ("rolling" search horizon).

Online Evolutionary Planning (OEP) [28], [7] is a recent evolutionary approach that is applicable to adversarial multi-action games. It optimizes only the action sequence of the current turn, without lookahead to future turns of the player or the opponent. It can therefore be seen as doing one iteration of RHEA at the beginning of each turn, and with a search horizon of one turn. The best action sequence found is then executed without "rolling" the horizon forward action by action.

OEP begins its search by creating an initial population of genomes, each genome representing a complete turn (fixed-length sequence of actions). Vanilla OEP chooses each of these genomes by repeatedly selecting random actions starting from the current game state. This population is then improved from

generation to generation, until a given computation time runs out. Each generation consists of the following four phases:

1. All genomes are translated to their respective phenotypes, the game states resulting from applying their action sequence to the current game state. The fitness of these phenotypes is then evaluated with the help of a static heuristic evaluation.

2. The genomes with the lowest fitness are removed from the population. The proportion of genomes to be removed is a parameter called the *kill rate*.

3. The surviving genomes are each paired with a randomly chosen different genome, and create an offspring through uniform crossover. If this crossover operator leads to an illegal action in the offspring, it is repaired by replacement with an action from the other parent, or otherwise with a random legal action.

4. A proportion of the offspring, determined by a parameter called the *mutation rate*, undergoes mutation. One randomly chosen action of the sequence is changed to another action randomly chosen from all legal actions. If this leads to illegal actions later in the sequence, they are replaced with random legal actions as well.

When the time budget is exhausted, OEP returns the action sequence represented by the current best genome, which is then executed action by action. In the words of Wang et al. "the action selection problem is seen as an optimization problem rather than a planning problem" [29]. This is currently the best-performing approach for turn-based multi-action adversarial games, in particular the test domain of this paper: Hero Academy [7]. It has also been applied to other problems such as micro battles [29] or online build order adaptation [30] in RTS games.

We are using the original OEP, as well as a new improved variant, as baselines in our experiments. The proposed EMCTS is similar to OEP in the sense that in multi-action adversarial games, it also searches a space of complete turns, which are connected to each other through the same mutation operator. It is different in being a tree search algorithm.

C. Hybrids of tree search and evolution

Several other methods have been published that combine ideas from tree search algorithms and evolutionary algorithms.

Gaina et al. [31] experimented in General Video Game AI (GVGAI) with splitting the total search time in two, using MCTS in the first half to provide an initial solution, which is then refined by RHEA in the second half. This was able to outperform RHEA, but not MCTS. Horn et al. [32] hybridized MCTS and RHEA in two different ways: By making use of limited-depth Monte Carlo simulations in the evaluation of RHEA genomes, and by running RHEA and MCTS separately and choosing the best solution found by either of them for execution. EMCTS on the other hand uses a single search algorithm, and a tree search with static state evaluation instead of an evolutionary search with rollouts for evaluation. Lucas et al. [33] used an evolutionary algorithm to improve the rollout policy of MCTS while the search is running. Perez-Liebana et al. [34] adapted a similar method for GVGAI, combining it with a knowledge base to improve reward calculations of given states. While improving MCTS or RHEA performance in various single-player games, the algorithms developed for the GVGAI framework are not straightforwardly applicable to multi-action adversarial games.

For adversarial games, Hong et al. [35] proposed a strategy to evolve paths through a game tree with the help of an evolutionary algorithm. While their approach assumes identical actions to be available in all states at the same search depth, which is not the case in most real-world games including our testbed Hero Academy, it gives an interesting indication for possible future work that could take opponent actions into account.

III. Methods

This section briefly describes the game we use as testbed, lists the search algorithms we are comparing to, and finally presents our approach: Evolutionary MCTS.

A. Test Domain: Hero Academy

Rules. Our test domain is a simplified[1] Java clone [36] of Hero Academy [37], a two-player turn-based tactics game. Players can use a variety of combat units, items, and spells by first drawing them from a card deck onto their hand, and then deploying, casting, or moving them on a battlefield of 9×5 squares. Special squares on this battlefield allow for unit deployment, boost the stats of individual units, or represent a player's two crystals. The game is won by the first player to either eliminate all enemy units, or to destroy both enemy crystals. More details on implementation and rules can be found in [28].

Fig. 1: The testbed game Hero AIcademy. The six symbols at the bottom represent the current player's hand, and the numbers below the doors represent the deck sizes. One of the red player's crystals has already been destroyed.

A central mechanic of the game are the *action points* (AP). For each turn, the player to move receives a number of action points – five in the standard form of the game. Each action

[1]For example, only the "Council" team of units is available.

point can be used for any one atomic action such as deploying a unit from the player's hand onto the battlefield, moving a unit on the battlefield, attacking an enemy unit, healing a friendly unit, and others. The player can spend any number of action points on a single unit, for example by moving it several times. With an average of 30-60 actions available per game state, depending on the playstyle, the full branching factor per turn can be roughly estimated to be $30^5 \approx 2.4 \times 10^6$ to $60^5 \approx 7.8 \times 10^8$. Finding the best sequence of actions for any given turn is therefore a challenging search problem in itself.

The order of cards in the deck as well as the opponent's cards are unknown to the Hero Academy player. However, this paper focuses on the challenge of multi-action turns, ignoring the aspects of hidden information and indeterminism as in [7].

In line with Justesen et al.'s prior work on Hero Academy, we use game knowledge for state evaluation as well as action pruning and ordering:

State evaluation. All algorithms compared in this paper use the same heuristic evaluation function. This function is a linear combination of features such as the current health of individual units, whether they are equipped with certain items, and whether they are standing on special squares. Improving this hand-coded function with machine learning, and testing if our conclusions still hold, could be worthwhile future work.

Action pruning and ordering. All algorithms compared in this paper use a form of hard pruning, removing a number of redundant or provably suboptimal actions from the set of available actions considered in any given state. The two MCTS variants considered as baselines also make use of static action ordering, giving the more promising actions priority in their expansion and rollout phases. The heuristics used for this are simpler and faster than those of the evaluation function.

The interested reader can refer to [28] for a full definition of the heuristic evaluation function and the pruning and ordering strategies.

B. Baseline Approaches

In order to make our results directly comparable to the literature, we are testing our approach against five of the algorithms described in [7]. Four of them are tree search techniques, and one is Online Evolutionary Planning representing the state of the art for Hero Academy.

Greedy Action. The Greedy Action AI chooses the first action of its turn with a simple one-ply search of all legal actions, maximizing the heuristic evaluation of the immediately resulting state. This is repeated for each action point, i.e. for all five actions of the turn.

Greedy Turn. The Greedy Turn AI chooses its actions by attempting a five-ply depth-first search of the entire turn, maximizing the heuristic evaluation of the leaf states resulting from full turns. It uses a transposition table in order to avoid re-visiting states. Actions are ordered for search with the evaluation function, which is especially important since Greedy Turn can usually not exhaustively search the entire turn in the given time limit.

Non-exploring MCTS. This AI is the first MCTS variant adapted for multi-action adversarial games in [7]. It searches a game tree as shown in Figure 2, in which each edge represents an additional action for the turn under consideration (or its application). The opponent's next turn can be reached by a tree deeper than five plies, the number of action points. The selection policy of this MCTS variant is UCB, and the rollout policy deterministically follows the action ordering heuristics. It was found to improve performance when rollouts are just long enough to complete the current turn of the player to act in the leaf node, calling the heuristic state evaluator at the end of the turn for a rollout result. The MCTS exploration factor is set to $C = 0$ in an attempt to grow a deep enough tree (pure exploitation).

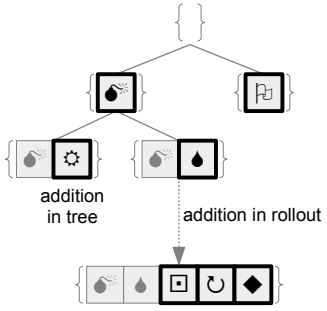


Fig. 2: Tree structure as searched by vanilla MCTS and its variants (non-expl. MCTS, BB-MCTS). Nodes represent partial action sequences, or the states resulting from them. Edges represent the addition of an atomic action to an action sequence, or the application of an atomic action to a state. After each node expansion, a rollout is performed for evaluation. (We use symbols to represent different atomic actions.)

Bridge-burning MCTS (BB-MCTS). This MCTS variant searches the same kind of tree shown in Figure 2. Instead of deterministic rollouts, it uses ϵ-greedy rollouts with $\epsilon = 0.5$, which also only reach to the end of the current turn of the leaf node. Its exploration factor is $C = 1/\sqrt{2}$. In order to grow a deep enough tree for multi-action turns however, it employs a technique called "bridge burning" in [7] – a re-invention of move-by-move search [8]. We are keeping the term "bridge burning" here, as the term "move" is ambiguous in Hero Academy, and also because we are going to generalize the concept of bridge burning to a different kind of tree in the next subsection.

The idea of BB-MCTS is to split the time budget for the current move search into five phases, equal to the number of actions per turn. During each phase, the MCTS search proceeds normally, but at the end of each phase, the most promising action at the root is executed, leading to the root state for the next phase. This can be implemented as the hard pruning strategy shown in Figure 3.

Online Evolutionary Planning. The OEP baseline is as described in Subsection II-B. In our experiments, we use the same parameter settings as suggested in [7]: A population size of 100, a kill rate of 0.5, a mutation rate of 0.1, and uniform crossover and mutation operators.

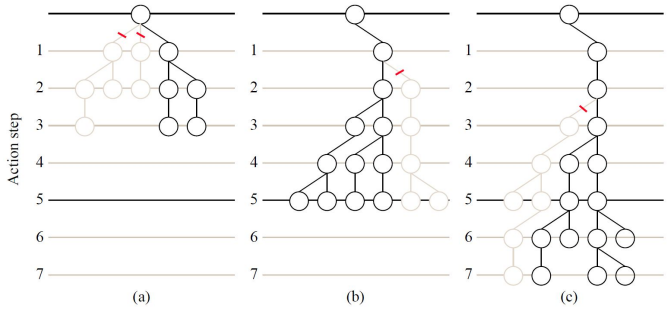


Fig. 3: The "bridge burning" search strategy (illustration adapted from [7]). (a) After phase 1, all branches but the best one are pruned at the root. (b,c) After phases 2, 3, ... n, pruning is applied at depth 2, 3, ... n. The partial tree below the best branch is retained.

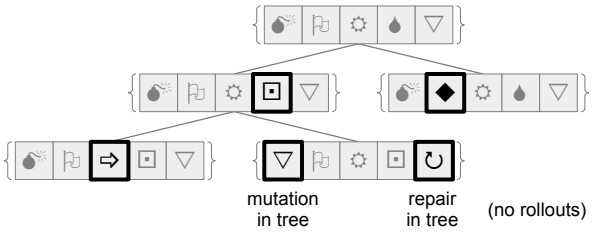


Fig. 4: Tree structure of Evolutionary MCTS. Nodes represent complete action sequences (genomes), or the states resulting from them. Edges represent the mutation of an atomic action within a genome. Repairs can be necessary if those mutations can lead to illegal genomes. After each node expansion, the evaluation function is called instead of a rollout. (We use symbols to represent different atomic actions.)

This algorithm is currently the best-performing approach for multi-action turn-based games such as Hero Academy. Although [7] shows it to be of similar strength to non-exploring MCTS and BB-MCTS in the standard form of the game with 5 action points per turn, OEP was shown to scale better to the tougher challenges of Hero Academy using 10 AP or more. Our experiments include those exponentially more complex variants as well.

C. Evolutionary MCTS

This subsection proposes our new search algorithm, *Evolutionary MCTS* or *EMCTS*, as applied to playing multi-action turn-based adversarial games. It combines the tree search of MCTS with the genome-based approach of evolutionary algorithms.

Instead of the vanilla MCTS tree seen in Figure 2, EMCTS builds a tree as shown in Figure 4. Instead of starting from an empty turn in the root, EMCTS starts from a complete sequence of five (or more, depending on the domain) actions – just like the genomes of OEP. Instead of growing a tree that adds one action to the current sequence with every edge, EMCTS grows a tree that mutates the current sequence with every edge – using the same mutation operator as OEP. And instead of using rollouts to complete the current turn and then evaluating it as our MCTS baselines do, we simply evaluate the solutions at the leaf nodes[2]. Backpropagation is unchanged.

EMCTS does not apply mutations randomly, but can choose exactly which action in the sequence to mutate and which other legal action to mutate it to[3]. While OEP turned the planning of the action sequence into an optimization problem, EMCTS thus takes the evolutionary optimization of the sequence and turns it back into a planning problem. It can be seen as tree search, but it can also be seen as a systematic exploration of the mutation landscape of OEP, giving evolution the benefit of lookahead.

Two questions need to be answered to fully flesh out EMCTS. First, where does the root sequence come from? EMCTS needs a *starting solution* to its search, just like EAs such as OEP need a starting population of solutions. Different approaches are possible – in this paper, we are using the Greedy Action algorithm described above for a quick and greedy initialization of the root. Second, what happens when a mutation leads to an illegal action sequence? We could filter these out by simulating every possible mutation in advance, but that would be computationally expensive. Instead, like OEP we are taking the classic evolutionary algorithm approach of using a *repair strategy* – in this paper, we are using the Greedy Action AI for repairs as well whenever necessary.

Note that the use of Greedy Action does not introduce additional heuristic knowledge, as all algorithms compared in this paper are working with the same evaluation function. However, we noticed that like EMCTS, OEP can also be significantly improved by using a Greedy Action repair policy instead of a random repair policy. This results in higher quality repairs on average. And just like EMCTS profits from a greedy root genome, OEP can profit from filling 20% of the starting population with Greedy Action sequences instead of random ones[4]. This kick-starts the search with higher-quality starting solutions. We are calling this new variant *greedy OEP* here, as opposed to *vanilla OEP* with random repairs and a purely random starting population as described in [28], [9], [7], and include it in our experiments for a fair comparison.

Finally, EMCTS results in an even larger branching factor than the vanilla MCTS variants. While the branching factor in Hero Academy games between the MCTS baselines was measured to be between 30 and 40, the branching factor of the mutation tree of EMCTS is about 30 *per action point* – so around 150 for the standard settings of the game with five action points. We found that an effective way of dealing with this is "bridge burning", just as applied to the regular MCTS tree by BB-MCTS. Instead of executing the most promising action at the root after every search phase like BB-MCTS, EMCTS executes the most promising mutation at the root after each phase. The number of bridge burning phases, of successive

[2]Evaluating at the leaf nodes is a well-known MCTS variant that was successfully employed for example in AlphaGo Zero and AlphaZero [3].

[3]No crossover operator is used.

[4]This performed better than filling 1%, 10%, and 50% of the starting population with Greedy Action sequences.

searches and prunings/mutations, is the only parameter of EMCTS we tuned (see the following section). The MCTS exploration factor was set to $C = 0$. The selection policy is UCB as in the other MCTS variants.

IV. Experiments and Results

This section describes our experimental setup for testing the proposed Evolutionary MCTS, as well as the results.

A. Experimental Setup

We tested EMCTS in Hero Academy against Greedy Action, Greedy Turn, non-exploring MCTS, BB-MCTS, and vanilla OEP as proposed in [7], as well as the improved greedy OEP as proposed in the previous section. All comparisons were performed on the standard settings of the game with 5 action points per turn, but also with altered rules allowing 10 AP or even 15 AP per turn[5]. This increases the complexity of a single turn exponentially, but gives a stronger indication of generalizability to other games which can have higher numbers of possible actions per turn. Furthermore, all comparisons were done at different time budgets of 200 ms per turn, 1 second per turn, and 5 seconds per turn. Each comparison consisted of 400 games, with EMCTS playing 200 games as the first player and 200 games as the second player. The map used is shown in Figure 1. Games that had no winner after 200 turns were counted as draws, i.e. half a win for each player.

All algorithms used the parameter settings described in Section III. The number of "bridge burning" phases for EMCTS was determined in preliminary experiments and set to 20 for 200 ms, 40 for 1 second, and 100 for 5 seconds per turn time controls. The number of phases for BB-MCTS were identical to the AP per turn, since it searches the type of tree shown in Figure 2 and does not profit from deeper searches. As no other algorithm was modified based on the AP per turn, EMCTS was also not specifically tuned for different AP.

B. Results

Table I shows the performance of the proposed Evolutionary MCTS against the five baselines and the improved greedy Online Evolutionary Planning.

EMCTS is significantly stronger than all baselines (Greedy Action, Greedy Turn, BB-MCTS, non-expl. MCTS, and vanilla OEP) at all time controls and all numbers of action points per turn. Its relative strength increases with the complexity of the search problem as measured in action points per turn. The newly proposed greedy OEP is a dramatic improvement over vanilla OEP as described in [7], but still significantly weaker than EMCTS at all action points at 200 ms per turn, and at all action points except for the lowest setting (5) at 1 s and 5 s per turn, where the two algorithms perform similarly. The results therefore show that Evolutionary MCTS is highly effective at a variety of time controls, and scales better with the complexity of the domain than all other tested approaches.

[5]20 or even 25 AP as in [7] were not included. As the authors noted, such high numbers of AP make it possible to win the game within very few turns, and make the winner very strongly depend on who gets the first turn. Strength differences between AIs are therefore harder to measure. More significant rule changes would have to be made to balance the game with such high AP.

Opponent	Action points per turn		
	5	10	15
200 ms per turn			
Greedy Action [7]	87.6%***	97.8%***	98.3%***
Greedy Turn [7]	96.9%***	100.0%***	100.0%***
BB-MCTS [7]	68.6%***	88.8%***	93.0%***
non-expl. MCTS [7]	74.5%***	91.8%***	92.0%***
vanilla OEP [7]	77.8%***	92.0%***	94.8%***
greedy OEP [this paper]	60.6%**	59.5%**	65.3%***
1000 ms per turn			
Greedy Action [7]	88.1%***	98.5%***	99.3%***
Greedy Turn [7]	92.8%***	99.0%***	100.0%***
BB-MCTS [7]	67.1%***	90.3%***	94.5%***
non-expl. MCTS [7]	65.5%***	93.5%***	97.3%***
vanilla OEP [7]	70.5%***	84.8%***	91.0%***
greedy OEP [this paper]	52.5%	58.8%*	61.8%***
5000 ms per turn			
Greedy Action [7]	91.9%***	99.0%***	99.8%***
Greedy Turn [7]	78.1%***	98.8%***	100.0%***
BB-MCTS [7]	67.0%***	90.3%***	94.8%***
non-expl. MCTS [7]	56.9%*	94.8%***	98.5%***
vanilla OEP [7]	69.0%***	80.3%***	87.5%***
greedy OEP [this paper]	51.4%	59.0%*	61.3%**

TABLE I: Win rates of EMCTS vs. all baselines at different time controls. 400 games per data point. Asterisks indicate significantly stronger play by EMCTS: $*p < 0.05$, $**p < 0.01$, $***p < 0.001$

Note that there is a tradeoff for "bridge burning" EMCTS between doing more phases (pruning all but the best mutation and continuing search from there), and having more time for each phase (to identify the best mutation). With search time, both the optimal number of phases as well as the optimal time per phase seem to increase. The settings found to perform best in our experiments have such high numbers of phases, and such little time for them, that EMCTS could be seen as a type of local search [38] or $(1, \lambda)$-Evolution Strategy [39]. At longer time settings though, deeper trees can form, and EMCTS turns into a new kind of genome-based planning, or evolution with lookahead. These connections are worth exploring more deeply in future work.

V. Conclusions and Future Work

This paper proposes a new algorithm for playing turn-based adversarial games, where each turn consists of a sequence of multiple actions. Such action sequences, common in strategy games, lead to the challenge of extremely large branching factors per turn. This is difficult to handle even for selective tree search methods such as MCTS, which typically search a tree of atomic actions, and specifically developed evolutionary algorithms such as OEP, which optimize entire action sequences.

Our new algorithm, called *Evolutionary MCTS* (EMCTS), is based on the idea of combining the tree search of MCTS with the sequence-based optimization of evolutionary algorithms. Instead of searching a vanilla MCTS tree, EMCTS searches a tree in which each edge mutates one action in a complete action sequence. Experiments on the game Hero Academy show that EMCTS convincingly outperforms several baselines from the literature, including the state of the art OEP and an improved variant of OEP introduced in this paper, at different time settings and numbers of actions per turn. EMCTS also scales better than any existing algorithm with the complexity of the problem. It is therefore the currently strongest algorithm for playing Hero Academy, and a promising candidate for other turn-based multi-action games such as Civilization, XCOM, Heroes of Might and Magic, or Into the Breach.

Several directions appear interesting for future work. First, the comparison between Evolutionary MCTS and the baseline algorithms could be deepened, including experiments with different initialization and repair strategies, different evaluation functions, more careful tuning of algorithm parameters such as OEP's population size, mutation rate, and kill rate, and possible improvements to MCTS methods such as stronger rollout policies. Second, various aspects of EMCTS could be considered in more detail, such as speed optimizations – it currently only evaluates roughly 20% as many action sequences per second as OEP. Mutations for expansion could for example be generated lazily in the tree nodes, and various MCTS enhancements could be used to improve their ordering. Third, the performance of EMCTS in other games could be tested, such as strategy games with longer matches and larger numbers of units. We are planning to apply it to Battle of Polytopia, a mobile turn-based strategy game in which armies can grow to 15 to 20 units or more in the late game. Unlike Hero Academy, Battle of Polytopia does not allow for any unit to move more than once per turn; however, additional complexity arises from units whose actions themselves consist of several atomic parts such as moving, attacking, and retreating. An interesting challenge for the application to commercial games is that the existence of a heuristic state evaluation function cannot generally be assumed, requiring machine learning approaches. Just like OEP, EMCTS could also be generalized to other problems such as micro battles [29] or online build order adaptation [30] in RTS games. In the former scenario, the genomes would consist of a list of scripts representing simple policies assigned to each unit, instead of a list of atomic actions for the player. In the latter scenario, the genomes would be candidate build orders, i.e. fixed-length sequences of future units and buildings to construct. Fourth, the problem of considering future actions of the opponent has not been tackled successfully yet, neither by OEP nor by EMCTS. Generalizing to larger classes of games will also require dealing with indeterminism and partial observability. And last but not least, the algorithmic similarities between Evolutionary MCTS and certain local search algorithms and evolutionary algorithms deserve further study, in order to further explore the idea of "evolution with lookahead".

REFERENCES

[1] L. Kocsis and C. Szepesvári, "Bandit based Monte-Carlo planning," in *17th European Conference on Machine Learning, ECML 2006*, ser. Lecture Notes in Computer Science, vol. 4212, 2006, pp. 282–293.

[2] R. Coulom, "Efficient selectivity and backup operators in Monte-Carlo Tree Search," in *5th International Conference on Computers and Games, CG 2006. Revised Papers*, ser. Lecture Notes in Computer Science, vol. 4630, 2007, pp. 72–83.

[3] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. P. Lillicrap, K. Simonyan, and D. Hassabis, "Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm," *CoRR*, vol. abs/1712.01815, 2017.

[4] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. Cowling, P. Rohlfshagen, S. Tavener, D. Perez-Liebana, S. Samothrakis, and S. Colton, "A survey of Monte Carlo Tree Search methods," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, pp. 1–43, 2012.

[5] P. I. Cowling, C. D. Ward, and E. J. Powley, "Ensemble Determinization in Monte Carlo Tree Search for the Imperfect Information Card Game Magic: The Gathering," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 4, pp. 241–257, 2012.

[6] T. Kozelek, "Methods of MCTS and the Game Arimaa," Master's thesis, Faculty of Mathematics and Physics, Charles University, Prague, 2009.

[7] N. Justesen, T. Mahlmann, S. Risi, and J. Togelius, "Playing Multi-Action Adversarial Games: Online Evolution versus Tree Search," *IEEE Transactions on Computational Intelligence and AI in Games*, 2017, in print.

[8] M. P. D. Schadd, M. H. M. Winands, M. J. W. Tak, and J. W. H. M. Uiterwijk, "Single-Player Monte-Carlo Tree Search for SameGame," *Knowledge-Based Systems*, vol. 34, pp. 3–11, 2012.

[9] N. Justesen, T. Mahlmann, and J. Togelius, "Online Evolution for Multi-action Adversarial Games," in *19th European Conference on Applications of Evolutionary Computation (EvoApplications 2016)*, ser. Lecture Notes in Computer Science, G. Squillero and P. Burelli, Eds., vol. 9597. Springer, 2016, pp. 590–603.

[10] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-Time Analysis of the Multiarmed Bandit Problem," *Machine Learning*, vol. 47, no. 2-3, pp. 235–256, 2002.

[11] S. Gelly and Y. Wang, "Exploration Exploitation in Go: UCT for Monte-Carlo Go," in *Neural Information Processing Systems Conference (NIPS), On-line trading of Exploration and Exploitation Workshop*, 2006.

[12] R. Coulom, "Computing elo ratings of move patterns in the game of Go," in *Computer Games Workshop*, 2007.

[13] G. M. J. B. Chaslot, M. H. M. Winands, J. v. d. Herik, J. W. H. M. Uiterwijk, and B. Bouzy, "Progressive Strategies for Monte-Carlo Tree Search," *New Mathematics and Natural Computation*, vol. 4, no. 03, pp. 343–357, 2008.

[14] D. Churchill and M. Buro, "Hierarchical Portfolio Search: Prismata's Robust AI Architecture for Games with Large Search Spaces," in *11th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE 2015*, A. Jhala and N. Sturtevant, Eds. AAAI Press, 2015, pp. 16–22.

[15] N. Justesen, B. Tillman, J. Togelius, and S. Risi, "Script- and cluster-based UCT for StarCraft," in *2014 IEEE Conference on Computational Intelligence and Games, CIG 2014*, 2014, pp. 1–8.

[16] D. Churchill and M. Buro, "Portfolio Greedy Search and Simulation for Large-scale Combat in StarCraft," in *IEEE Conference on Computational Intelligence in Games, CIG 2013*. IEEE, 2013, pp. 1–8.

[17] S. Ontañón, "The Combinatorial Multi-Armed Bandit Problem and Its Application to Real-Time Strategy Games," in *9th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE-13*, G. Sukthankar and I. Horswill, Eds. AAAI, 2013.

[18] T. Yee, V. Lisý, and M. H. Bowling, "Monte Carlo Tree Search in Continuous Action Spaces with Execution Uncertainty," in *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016*, S. Kambhampati, Ed. IJCAI/AAAI Press, 2016, pp. 690–697.

[19] S. M. Lucas and G. Kendall, "Evolutionary Computation and Games," *IEEE Computational Intelligence Magazine*, vol. 1, no. 1, pp. 10–18, 2006.

[20] S. Risi and J. Togelius, "Neuroevolution in Games: State of the Art and Open Challenges," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 9, no. 1, pp. 25–41, 2017.

[21] N. Cole, S. J. Louis, and C. Miles, "Using a genetic algorithm to tune first-person shooter bots," in *2004 Congress on Evolutionary Computation (CEC 2004)*, 2004, pp. 139–145.

[22] G. M. J. B. Chaslot, M. H. M. Winands, I. Szita, and H. J. van den Herik, "Cross-entropy for Monte-Carlo Tree Search," *ICGA Journal*, vol. 31, no. 3, pp. 145–156, 2008.

[23] A. M. Alhejali and S. M. Lucas, "Using genetic programming to evolve heuristics for a Monte Carlo Tree Search Ms Pac-Man agent," in *2013 IEEE Conference on Computational Intelligence and Games, CIG 2013*, 2013, pp. 1–8.

[24] A. Benbassat and M. Sipper, "Evomcts: A scalable approach for general game learning," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 6, no. 4, pp. 382–394, 2014.

[25] C. F. Sironi and M. H. M. Winands, "On-line parameter tuning for Monte-Carlo Tree Search in General Game Playing," in *Computer Games Workshop*, 2017, pp. 75–95.

[26] D. Perez-Liebana, S. Samothrakis, S. M. Lucas, and P. Rohlfshagen, "Rolling Horizon Evolution versus Tree Search for Navigation in Single-Player Real-Time Games," in *2013 Genetic and Evolutionary Computation Conference, GECCO '13*, C. Blum and E. Alba, Eds. ACM, 2013, pp. 351–358.

[27] R. D. Gaina, J. Liu, S. M. Lucas, and D. Perez-Liebana, "Analysis of Vanilla Rolling Horizon Evolution Parameters in General Video Game Playing," in *20th European Conference on Applications of Evolutionary Computation, EvoApplications 2017*, ser. Lecture Notes in Computer Science, G. Squillero and K. Sim, Eds., vol. 10199, 2017, pp. 418–434.

[28] N. Justesen, "Artificial Intelligence for Hero Academy," Master's thesis, IT University of Copenhagen, 2015.

[29] C. Wang, P. Chen, Y. Li, C. Holmgard, and J. Togelius, "Portfolio Online Evolution in StarCraft," in *Twelfth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE-16*, 2016, pp. 114–120.

[30] N. Justesen and S. Risi, "Continual Online Evolutionary Planning for In-game Build Order Adaptation in StarCraft," in *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2017*, P. A. N. Bosman, Ed. ACM, 2017, pp. 187–194.

[31] R. D. Gaina, S. M. Lucas, and D. Perez-Liebana, "Population Seeding Techniques for Rolling Horizon Evolution in General Video Game Playing," in *2017 IEEE Congress on Evolutionary Computation, CEC 2017*. IEEE, 2017, pp. 1956–1963.

[32] H. Horn, V. Volz, D. P. Liebana, and M. Preuss, "MCTS/EA Hybrid GVGAI Players and Game Difficulty Estimation," in *IEEE Conference on Computational Intelligence and Games, CIG 2016*. IEEE, 2016, pp. 1–8.

[33] S. M. Lucas, S. Samothrakis, and D. Perez-Liebana, "Fast Evolutionary Adaptation for Monte Carlo Tree Search," in *17th European Conference on Applications of Evolutionary Computation, EvoApplications 2014*, ser. Lecture Notes in Computer Science, A. I. Esparcia-Alcázar and A. M. Mora, Eds., vol. 8602. Springer, 2014, pp. 349–360.

[34] D. Perez-Liebana, S. Samothrakis, and S. M. Lucas, "Knowledge-based Fast Evolutionary MCTS for General Video Game Playing," in *2014 IEEE Conference on Computational Intelligence and Games, CIG 2014*, 2014, pp. 1–8.

[35] T. Hong, K. Huang, and W. Lin, "Adversarial Search by Evolutionary Computation," *Evolutionary Computation*, vol. 9, no. 3, pp. 371–385, 2001.

[36] Niels Justesen, "Hero AIcademy." [Online]. Available: https://github.com/njustesen/hero-aicademy

[37] Robot Entertainment, "Hero Academy." [Online]. Available: http://www.robotentertainment.com/games/heroacademy/

[38] H. H. Hoos and T. Stützle, *Stochastic Local Search: Foundations & Applications*. Elsevier / Morgan Kaufmann, 2004.

[39] H. Beyer and H. Schwefel, "Evolution Strategies - A Comprehensive Introduction," *Natural Computing*, vol. 1, no. 1, pp. 3–52, 2002.

Skilled Experience Catalogue: A Skill-Balancing Mechanism for Non-Player Characters using Reinforcement Learning

Frank G. Glavin
School of Computer Science,
National University of Ireland, Galway.
frank.glavin@nuigalway.ie

Michael G. Madden
School of Computer Science,
National University of Ireland, Galway.
michael.madden@nuigalway.ie

***Abstract*—In this paper, we introduce a skill-balancing mechanism for adversarial non-player characters (NPCs), called *Skilled Experience Catalogue* (SEC). The objective of this mechanism is to approximately match the skill level of an NPC to an opponent in real-time. We test the technique in the context of a First-Person Shooter (FPS) game. Specifically, the technique adjusts a reinforcement learning NPC's proficiency with a weapon based on its current performance against an opponent. Firstly, a catalogue of experience, in the form of stored learning policies, is built up by playing a series of training games. Once the NPC has been sufficiently trained, the catalogue acts as a timeline of experience with incremental knowledge milestones in the form of stored learning policies. If the NPC is performing poorly, it can jump to a later stage in the learning timeline to be equipped with more informed decision-making. Likewise, if it is performing significantly better than the opponent, it will jump to an earlier stage. The NPC continues to learn in real-time using reinforcement learning but its policy is adjusted, as required, by loading the most suitable milestones for the current circumstances.**

***Index Terms*—Skill balancing, Dynamic Difficulty Adjustment, reinforcement learning, First Person Shooter**

I. Introduction

This paper presents a new mechanism for Dynamic Difficulty Adjustment in the context of reinforcement learning called Skilled Experience Catalogue (SEC). Specifically, we store the current policy of the non-player character (NPC) at various intervals during an initial training phase. Once the training phase is complete, the base policy of the NPC can be adjusted in real-time, influenced by a threshold value, to approximately match the skill level of the current opponent.

To test our SEC mechanism, we apply it to the weapon proficiency of an NPC bot in a First-Person Shooter (FPS) *Deathmatch* game. Specifically, the mechanism applies only to the learned task of aiming/firing a weapon at an enemy. The NPC has fixed strategies for the other in-game tasks such as item collection, opponent evasion and travelling around the map. The NPC bot is initially trained against a single opponent and builds up a catalogue of reinforcement learning policies as it gains experience from using the weapon over time. These stored policies, that loosely represent skill level, can then be loaded in subsequent games to balance the gameplay against the current opponent. We demonstrate this SEC mechanism against five different levels of fixed-strategy opponents and show that a single catalogue of experience can be used to closely match the performance of each. The NPC that we have developed is an adversarial one [1] which contrasts with supportive companion NPCs [2] found in some game genres.

The approach that we present is novel in that it is using a by-product of the bot's learning process to create *milestones* which represent the knowledge acquired at the different stages of learning. The policies of the bot are stored, at different stages, to keep a sequential catalogue of experience. The bot can then jump to the most appropriate policy to coincide with the skill level of the current opponent while continuing to adapt based on its in-game experience. Our approach does not require manually optimising parameters to represent different skill levels. Conversely, we are sampling from the natural learning progress of the agent over time.

II. Background Information

A. Dynamic Difficulty Adjustment

Traditionally, human computer game players select a difficulty setting from a menu before beginning the game. This can be as simple as selecting *easy* / *medium* / *hard* or can include more detailed options for the player to choose from. These settings have a direct, and usually static, effect on the skill level of the NPCs. Such fixed-difficulty settings can often be too broad. For instance, a setting that is intended to be easy may nonetheless be too difficult for some players. Players may also improve their performance at different rates. The traditional approach does not make use of the player's current performance measures to direct the gameplay. While this approach has the benefit of simplicity, from a game development point of view, it can lead to predictable gameplay when static rule-based opponents are deployed which can adversely affect the entertainment value of the game. Dynamic Difficulty Adjustment (DDA) [3], which can also be referred to as Dynamic Game Balancing (DGB) [4], involves identifying the player's performance and skill level, and then dynamically adjusting the difficulty level accordingly. The goal of this is

978-1-5386-4359-4/18/$31.00 ©2018 IEEE

to ensure that the game remains challenging and can cater for many different players of varying skill levels.

B. Reinforcement Learning

Reinforcement learning (RL) is a branch of artificial intelligence in which a learner, often called an *agent*, interacts with an environment to achieve an explicit goal or goals [5]. The environment consists of a set of states, called the *state space*, and the agent must choose an available action from the *action space* when in a given state at each time step. The agent learns from its interactions with the environment, receiving feedback for its actions in the form of numerical rewards, and aims to maximise the reward values that it receives over time. Two common approaches to storing/representing a *policy* in reinforcement learning are *generalisation* and *tabular*. With generalisation, a function approximator is used to generalise a mapping of states to actions. The tabular approach, which is used in our research, stores numerical representations of all state-action pairs in a lookup table. The agent's decision-making involves choosing between exploring the effects of taking novel actions and exploiting the knowledge that it has acquired from earlier exploration. Reinforcement learning is inspired by the process by which humans interact with the world and learn from experience.

C. Game Environment and Development Tools

For this research, we use the game Unreal Tournament 2004 (UT2004) which is a commercial FPS game [6]. The agents are developed using a toolkit called *Pogamut 3* [7] which is an open-source development platform for creating virtual agents in the 3D game environment of UT2004. The main objective of Pogamut 3 is to simplify the coding of actions taken in the environment, such as path finding, by providing a modular development platform. Making use of these primitives, the focus of our development is on producing intelligent NPC behaviour.

III. Motivation

When computer-controlled opposition is too strong, human players can become frustrated with the gameplay. Conversely, opponents that are too weak result in predictable games in which human players do not feel challenged [8]. The challenge of a game is widely considered to play a crucial role in the player's overall enjoyment [9]. We believe that successful game AI requires techniques to be developed in which the NPCs can learn good tactics independently as well as being both unpredictable and adaptive to their surroundings. Keeping a player's win and loss rate close and unpredictable in a game can increase the player's overall suspense and the game's outcome uncertainty. Abuhamdeh *et al.* [10] carried out a study on the relevance of outcome uncertainty and suspense for intrinsic motivation and concluded that games with higher outcome uncertainty were more enjoyable to play.

We observe that modern computer games can often lack flexibility with regards to difficulty settings and this can lead to mismatches between the player's ability and the overall difficulty of the game. DDA can be used to ease the learning process for beginners. Difficulty settings are balanced in real-time in contrast to traditional approaches that involve extensive user testing and redesign in order to identify suitable levels. This can be a costly and time-consuming process [11].

IV. Related Research

Hunicke and Chapman [3] presented an interactive DDA system called *Hamlet* which is an integrated set of libraries within the *Half-Life* SDK. The Hamlet system has an *evaluation function*, which maps the current state of the game world to an evaluation of the player's performance and an *adjustment policy*, for mapping the evaluation to adjustments in the game world. Hamlet monitors incoming game data and estimates the player's future state from the data. If an undesirable state is predicted, the system will intervene and adjusts the game settings as required. Hunicke [12] used Hamlet to examine the requirements for incorporating effective dynamic difficulty adjustment into an FPS game. The aim of the study was to identify if DDA could be performed effectively, without degrading the core gameplay experience for the user. The authors reported that their preliminary results show an improvement in player performance, while retaining the player's sense of agency and accomplishment.

Spronck *et al.* [13] showed the extent to which their technique of *dynamic scripting* [14] could be used to adapt game AI to balance the gameplay in a simulation closely related to the Role-Playing Game (RPG) *Baldur's Gate*. The authors focussed on enhancing the difficulty-scaling properties of the dynamic scripting technique. These were high-fitness penalising, weight clipping, and top culling. The *reward peak value* in dynamic scripting determines how effective the opponent behaviour will be. With high-fitness penalising, this value is adjusted after every fight depending on the outcome. If the computer-controlled opponent wins, it is reduced; otherwise it is increased. There is also a maximum and minimum value that this reward peak value can be. The *maximum weight value* determines the maximum level of optimisation a learned tactic can achieve. With weight clipping, this value is automatically changed to balance the overall gameplay. Top culling is similar to weight clipping, however, rules with a weight greater than the maximum weight value are allowed. Those that exceed the maximum weight value will not be selected for a generated script which will force the computer-controlled opponent to use weaker tactics. The authors reported that, of the three different difficulty-scaling enhancements, the top-culling enhancement was the best choice. It was reported that it produced results with low variance, was easily implemented, and was the only one of the three enhancements that managed to force a balanced game when inferior tactics were used.

Tan *et al.* [15] presented two adaptive algorithms, based on ideas from reinforcement learning and evolutionary computation, to scale the difficulty of the game AI to improve player satisfaction. They introduced two controllers, namely, the adaptive uni-chromosome controller (AUC) and the adaptive

duo-chromosome controller (ADC). The authors examined the effects of varying the learning and mutation rates and proposed general rules for setting these parameters. The authors carried out experimentation using a modified version of the car simulator used in the Simulated Car Racing Competition held during the 2007 IEEE Congress on Evolutionary Computation (CEC 2007). It was demonstrated that the proposed algorithms can match their opponents in terms of mean scores and winning percentages. The authors also reported that both algorithms were able to generalize well to a variety of opponents.

Vicencio-Moriera *et al.* [16] carried out three separate studies on the performance of different *aim assist* techniques in an FPS game developed by the authors using the Unreal Development Kit (UDK). Aim assistance in FPS games is used to make it easier for players to select on-screen targets. This enables players with less skill to perform at a more competitive level increasing the competition and, in turn, the enjoyment of the player, which essentially balances the gameplay. The authors compared the following aim assist techniques: *target lock* (moves the crosshairs of the player's weapon to the closest target's head), *bullet magnetism* (bullets towards the closest target if within the activation range), *area cursor* (physical size of the crosshair changes), *sticky targets* (changes control-to-display ratio when the crosshairs are over a target) and *gravity* (targets have an attractive force dragging crosshairs towards them). The authors reported that the assists worked well in a target-range scenario, but their performance was reduced when real-game elements were introduced. They reported that bullet magnetism and area cursor worked well in a wide variety of situations but techniques such as target lock, while working well, were too perceptible to be successful for balancing. Vicencio-Moriera extended this work, in [17], by refining the bullet magnetism and area cursor techniques. They also developed a new technique to maintain the effect of the aim assist for a longer duration. They also included a map for novice players that shows them the location of expert players and incorporated different rates of damage for stronger and weaker players. The authors report that the new balancing schemes were extremely effective with the ability to balance the gameplay amongst players with large skill differences.

Our approach, which is outlined in the following section, differs from the aforementioned methods in that it uses real-time learning during the game coupled with a mechanism for loading policies that closely match the current opponent.

V. Skilled Experience Catalogue

We designed SEC based on the premise that there is a progressive timeline which begins with poor performances and ends with good performances as the agent learns how to perform a task over time. In general, SEC involves storing the policy of an RL agent at periodic intervals during an initial training phase. The agent will begin the training phase with no knowledge of the environment or intuition on what the best actions are to take given the circumstances. Well-designed learning agents will depict a clear upward trend in performance over time as the agent gains more experience. This time-based increase in performance is crucial to the success of using SEC. Milestones of the timeline of experience are stored by recording the RL policy at set intervals during the training phase. Once the experience catalogue has been populated with these learning milestones they can then be loaded to either increase or decrease the current underlying ability of the agent by effectively adding or removing experience (see Figure 3). Our application of SEC is concerned with balancing the *Assault Rifle* skill of a Deathmatch NPC playing against a single opponent. The NPC is initially trained using the shooter bot implementation from Glavin and Madden [18][19].

The main features of this implementation are as follows.

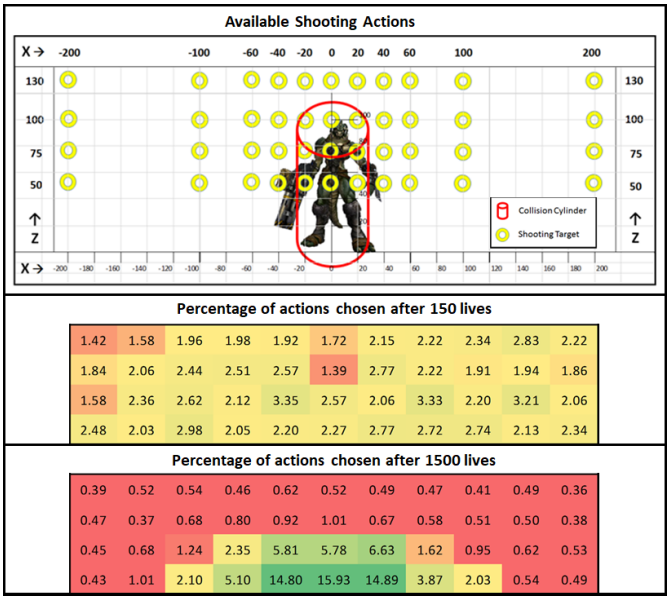


Fig. 1. Shooting actions available (top) and varied decision-making based on experience [19] (middle/bottom). The heat maps represent the percentage of time that a specific action was chosen up to that point. This is illustrated for those chosen up to 150 lives and up to 1500 lives respectively to show the improved decision making.

The state space is made up of a series of discretised features including the relative speed, relative moving direction, relative rotation, and distance to the opponent. The velocity and direction values of the opponent are translated into the NPC's point of view so that the NPC's actions can include discretized variations of shooting in a forward direction. The opponent's direction and speed are recorded relative to the NPCs own speed and from the perspective of the NPC looking directly ahead. Full details on the state space can be found in Glavin and Madden [19].

Figure 1 shows the actions that are available to the NPC, the percentage of those chosen after 150 lives and the percentage of those chosen after 1500 lives. These values were chosen as representative early and late stage learning to highlight the differences in decision-making as the NPC gains experience. The actions are expressed as different target directions in which the NPC can shoot, and which are skewed from the opponent's absolute location on the map. The amount of

skew along the X-axis (left and right) and Z-axis (up and down) varies by different fixed amounts as shown in the upper segment of Figure 1. These actions were designed specifically for the Assault Rifle weapon. The logic of this shooter NPC is based on the SARSA(λ) [5][20] algorithm which uses eligibility traces [21] to speed up learning by allowing past actions to benefit from the current reward. The NPC receives a reward of 250 every time the system records that it has caused damage to the opponent with the shooting action. If the NPC shoots the weapon and does not hit the opponent, it receives of penalty of -1. We use a tabular approach to represent the policy of the learner by storing numerical representations (Q-values) of all state-action pairs in a lookup table (Q-table). These are periodically stored and used to represent base learning levels (policies) as milestones. Full details of this algorithm and NPC implementation can be found in Glavin [22]. The following sections include details on the training experimentation and milestone switching mechanism for SEC.

A. Training Experience

The NPC must play a series of games against an opponent to populate the catalogue of experience. Each time the NPC dies it will write out its Q-table to a file. Once this training period has been completed, milestone Q-tables (from set intervals of the learning timeline) must be selected and placed into the SEC. We have chosen 100 deaths to represent a milestone in our implementation. That is to say that a milestone Q-table is added to the SEC after 100 deaths, 200 deaths, 300 deaths and so on. Since continuous reinforcement learning is being used, we believe that the resulting performance will be more natural and will make the adjustments of the skill level harder to detect.

B. Skill Adjustments

Once the SEC has been populated with the desired number of milestones, the mechanism for changing milestones must be set. Our implementation uses a simple positive/negative threshold for kill-death differences (KDDs) to drive the switching between milestones. Specifically, if the KDD between the NPC and the opponent exceeds a value of 5 (meaning the bot is performing significantly better), the policy will revert back to the previous milestone from the SEC. It will continue to step back through the milestones, after every opponent death, while the KDD remains above 5 or until it has reached the first milestone (empty Q-table; no knowledge). If the KDD falls below -5, the next milestone will be chosen from the SEC after every death until either the KDD returns to the *match range* ($-5 \geqslant KDD \leqslant 5$) or the highest milestone has been reached. The current milestone will remain unchanged while the KDD value is within the match range. The value 5 was chosen after a series of preliminary runs. This value could be increased to make the skill adjustments less prominent or reduced to enable faster skill adjustments.

The learning algorithm that is controlling the bot is as described in Glavin and Madden [19]. Both Periodic Cluster Weighted Rewarding (PCWR) and Persistent Action Selection (PAS) are both enabled, and all of the algorithm settings are as described in that research work.

VI. Experimentation

In this section, we outline how we first trained the NPC by playing against a single opponent for 100 individual thirty-minute games. The purpose of this training phase is to populate the catalogue of experience as the NPC acquires knowledge through trial and error over time. After this, we discuss the experiments in which the NPC used the SEC mechanism, while playing against opponents with differing skill levels, to balance the game play. We then discuss re-running these experiments with the technique disabled to carry out a comparative analysis.

A. Training Experiments

There are eight different pre-programmed native bot skill levels in UT2004 that are designed to increase the challenge for human players as the skill level is increased. High-level descriptions of the attributes associated with the first five of these skill levels, as reported in [23], are listed in Table I. The first step of our experimentation involved training the NPC by playing it against a Level 5 (Adept) opponent. We ran one hundred 1-vs-1 Deathmatch (30 minute) games on the Training Day map. This is a small map which encourages almost constant combat. The resulting data showed that the learner NPC performed poorly during the early games in which it died much more often than killing the opponent. This would

Skill	Attributes
Novice	60% of regular running speed, will not move during combat unless very weak, limited perception with 30° field of view, shooting aim can range 30° off target, slow to turn.
Average	70% of regular running speed, slightly higher shooting accuracy, turns slightly faster than novice.
Experienced	80% of regular running speed, will move and fire simultaneously, 40° field of view, can turn by more than $^1/_2$ per second.
Skilled	90% of regular running speed, can double jump, 60° field of view, turns more than $^5/_8$ per second.
Adept	Run at full speed, will dodge enemy fire, will close in on enemy, aim "leads" the target, 80° field of view, turn almost $^3/_4$ per second.

TABLE I
UT2004 native bot skill attributes. (Source: [23])

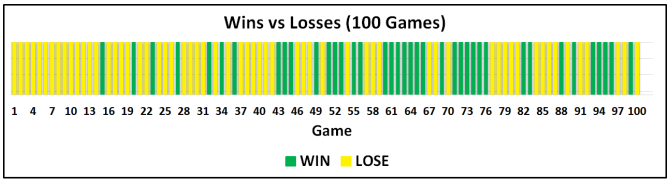


Fig. 2. Games won and lost during the one hundred 30 minute training games. The number of games won increases in the latter stages of the training phase.

be expected as the NPC needs time to experience all the game states and experiment with the different actions. The NPC began to outperform the opponent at the half way point of the 100 games. Figure 2 shows the number of wins and losses that the NPC recorded against the Level 5 opponent during the 100 training games. The NPC must play 15 games before it manages to defeat the Level 5 opponent. The NPC manages to beat the opponent much more frequently when it passes the half way point of the 100 games. This verifies that the NPC is in fact learning how to play more proficiently as it gathers experience. The NPC wins 39 out of the 100 games with 28 of these wins occurring in the second half of the games played. During the 100 games, the NPC died almost 9000 times. As a result of this we stored 90 milestone Q-tables. These were from no experience (empty Q-table) up to having died 8900 times in increments of 100.

Table II compares the average kills, deaths and kill-death (KD) ratio from the first 25 games (First Quarter - Q1) to the last 25 games (Fourth Quarter - Q4) in order to test if the enabled learning is leading to a statistically significant improvement in the average performance in the latter stages of learning. The ** in the table entries signifies that there are statistical differences with significance level α equal to 0.05 using an unpaired two-tailed t-test. It is important to note that in each successive game the bot begins with the knowledge that it has built up from all of the previous games. Therefore, the examples are not strictly independent as memory from each game is persisting over time. The settings of the individual 30-minute games and the opponents remain consistent throughout the 100 games. Both sample sets comprise individual games that take place in a period of 12.5 hours of learning.

The purpose of this comparison is to check for a statistically significant difference in the average performance between these two learning periods (early and later learning) in order to verify that the bot continues to improve its performance over time. For instance, if there was no significant difference between the bot's average performance during the period of 0 to 12.5 hours and the bot's average performance during the period of 37.5 to 50 hours then we could conclude that the effect of learning on the bot's performance had already plateaued during the earlier stages of learning. Table II shows that this is not the case and that the average number of kills achieved, and the average kill-death ratio has improved, at a 95% confidence level, in the later learning period. We have also confirmed from the bot logs that the bot continues to encounter new states throughout all of the training games and therefore may have an opportunity to learn a better policy in the latter stages of the training phase.

B. SEC-Bot versus Fixed Strategy Opponents

From here on, we will refer to the NPC that uses the SEC as *SEC-Bot*. The SEC-Bot begins each new game that it plays with no experience and then increases or decreases its knowledge as required, based on the threshold, by using the Q-table milestones. These experiments were also carried out on the Training Day map.

TABLE II
COMPARISON OF PERFORMANCE BETWEEN Q1 AND Q4. LEVEL OF CONFIDENCE: ** = 95%. THIS SHOWS AN INCREASE IN PERFORMANCE IN THE LATTER STAGES OF LEARNING.

	Q1: Games 1 to 25 Avg (Std Error)	**Q4: Games 75 to 100 Avg (Std Error)**
Kills	79.20 (± 2.09)	**84.56 (± 1.46) ****
Deaths	91.60 (± 2.02)	87.52 (± 1.46)
KD Ratio	0.88 (± 0.04)	**0.98 (± 0.03) ****

Table III shows the number of times that the SEC-Bot won, lost and drew against the first 5 levels of the native fixed-strategy bots. The SEC-Bot won 39, lost 30 and drew 6 of the 75 games that were played (15 individual games per level of native opponent) which shows that there is a large amount of outcome uncertainty when playing against five different levels of opponent. For instance, the SEC-Bot does not constantly beat the Level 1 opponent all the time nor does it struggle to play at the same standard as the Level 5 opponent. We can see from the table that the win and loss rate for the SEC-Bot are closely matched and that it is managing to successfully balance the gameplay against the different levels of opponent using just a single catalogue of experience. As we will see later, the difference between a win and a loss is often only a small number of kills. The following figures, from Figure 4 to Figure 8, show the results of the SEC-Bot playing a total of 15 thirty-minute games against five different levels of opponent. The purpose of this is to observe how well it can match the opponent's skill level, with respect to killing and being killed, by using the SEC mechanism. From the results, we can see that the SEC-Bot manages to closely match the kill rate of the opponent over each of the different levels. During these experiments, we noted the changes to the milestones that were occurring during the game-play. These include any time the bot moved up or down a milestone from the SEC and are listed as *milestone adjustments* on the figures. A *policy clearance* occurs when the SEC-Bot is currently using Policy 0 and still out-performing the opponent. It involves clearing all of the Q-values that were built up during gameplay and essentially re-starting learning. For Level 1, the SEC-Bot never increased to a higher milestone than Policy 0. At the beginning of each

TABLE III
WINS, LOSSES AND DRAWS FOR THE SEC-BOT AGAINST EACH OF THE 5 LEVELS.

Opponent Skill Level	**Win**	**Lose**	**Draw**
Level 1	8	6	1
Level 2	9	6	0
Level 3	8	4	3
Level 4	7	6	2
Level 5	7	8	0

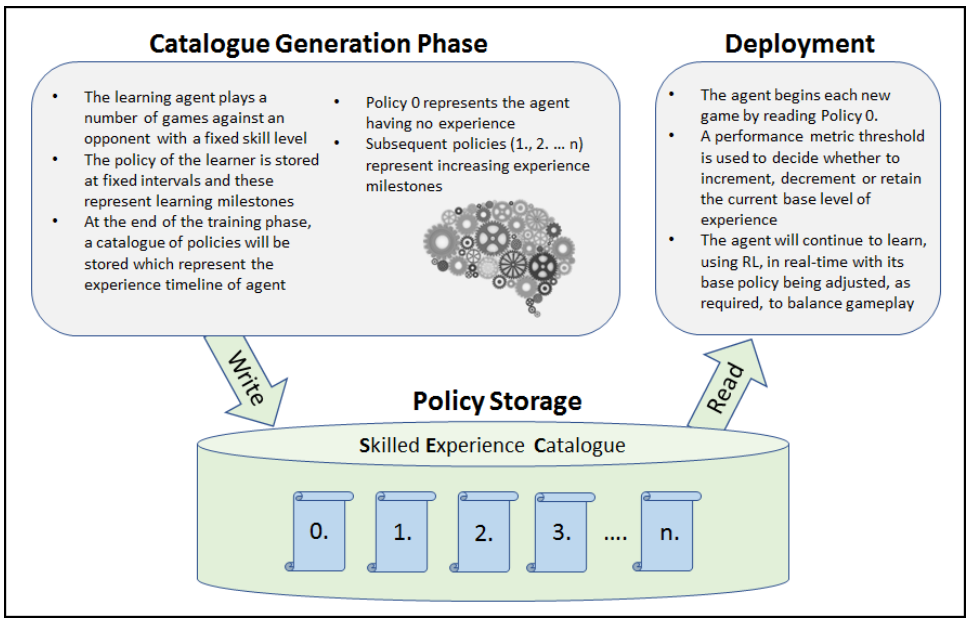


Fig. 3. An overview of Skilled Experience Catalogue.

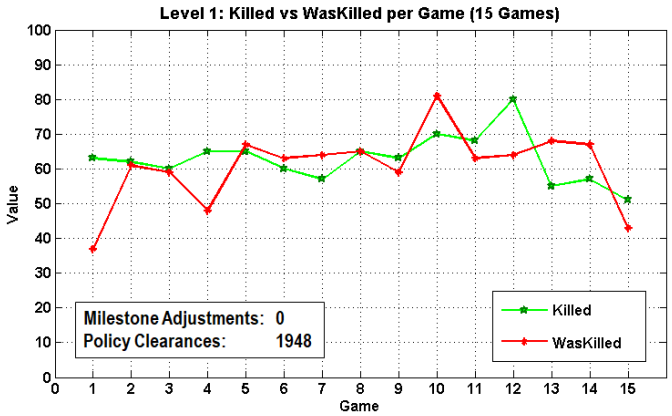


Fig. 4. SEC-Bot: Killed vs WasKilled against a Level 1 opponent.

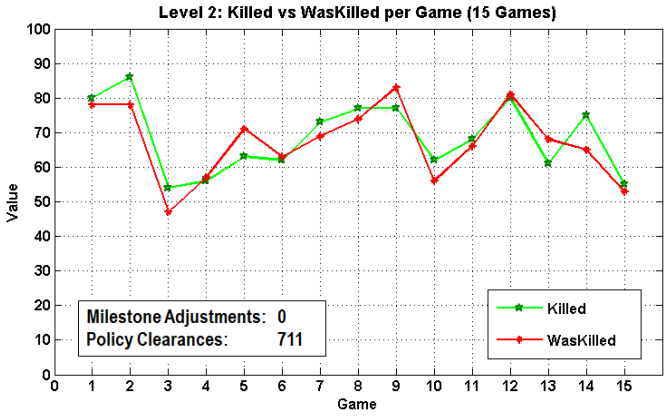


Fig. 5. SEC-Bot: Killed vs WasKilled against a Level 2 opponent.

game, it started with no knowledge and then only required the in-game learning experience to remain balanced with this opponent. It did, however, have to carry out a policy clearance over 1900 times throughout the games as it was out-performing the Level 1 opponent at the early stages of learning. The SEC-Bot, once again, had no milestone increases during the Level 2 games whereas the changes were much more prevalent against the more difficult opponents (Level 3, Level 4, and Level 5). The SEC-Bot had to almost immediately rise up through the milestones (once it fell below the threshold), before stopping at the highest point, while playing against the Level 5 opponent. This explains the larger variance that is seen in Figure 8 compared to the other results. It is important to recall that the SEC-Bot was initially trained using this level of opponent and it took almost 50 games before it had enough knowledge and managed to start convincingly defeating the opponent. Even at this point, it was still losing some games.

C. RL-Bot versus Fixed Strategy Opponents

We also ran all the above experimentation with the SEC mechanism disabled to perform a comparative analysis. The results, recorded over the 15 thirty-minute games for each skill level, are shown in Table IV. The *RL Only* entries represent the reinforcement learning bot with the SEC mechanism disabled. It therefore continues to learn as it gains experience and does not attempt to match the level of the opponent. For each level of opponent, we report the final KD ratio and the number of accumulated kills and deaths over the 15 thirty-minute games for the NPC. From the results, we can see that the performance of the RL Only bot depends on the skill of the opposition whereas the SEC-Bot can adjust its performance to

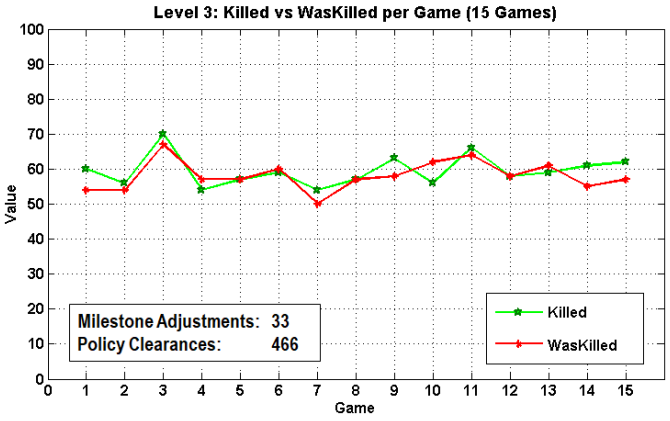


Fig. 6. SEC-Bot: Killed vs WasKilled against a Level 3 opponent.

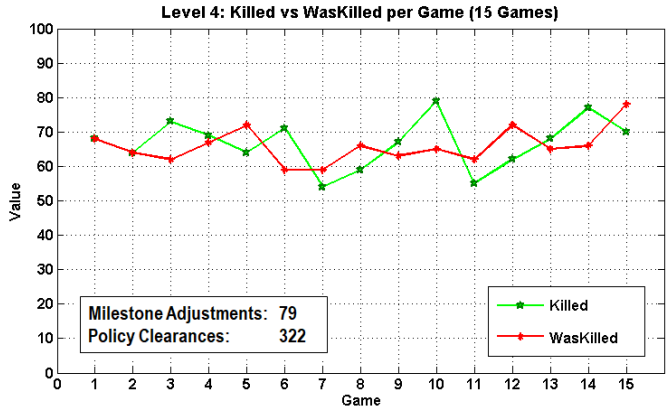


Fig. 7. SEC-Bot: Killed vs WasKilled against a Level 4 opponent.

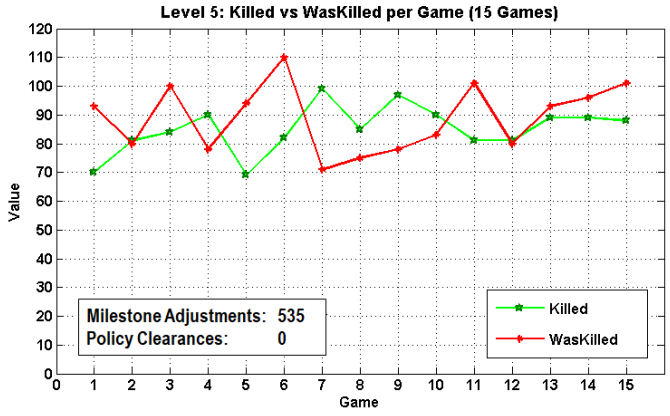


Fig. 8. SEC-Bot: Killed vs WasKilled against a Level 5 opponent.

retain a KD ratio of approximately 1.0 in each case. Figure 9 shows a comparative visualisation of the KD ratio for the SEC-Bot versus the RL Only bot. The ratio is plotted for each game *incident* (an incident occurs when the NPC has killed an opponent or died itself) against the Level 3 opponent. This

TABLE IV
COMPARISON OF PERFORMANCE WHEN ENABLING AND DISABLING THE SEC MECHANISM

Level	RL Only KD Ratio	SEC-Bot KD Ratio	RL Only Kills|Deaths	SEC-Bot Kills|Deaths
1	3.90	1.01	1955 | 501	988 | 974
2	3.73	1.03	1915 | 514	976 | 944
3	2.46	1.00	1561 | 635	1014 | 1011
4	1.33	1.00	1214 | 915	1004 | 1005
5	0.70	0.99	1032 | 1476	1302 | 1311

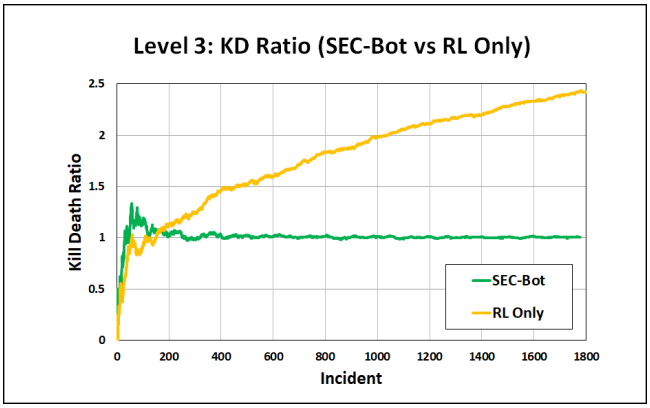


Fig. 9. KD Ratio when the SEC mechanism is enabled versus disabled.

clearly shows that SEC achieves successful game balancing with an approximate 1:1 KD ratio over time.

VII. DISCUSSION

The success of our approach relies on the implicit assumption that, during the training phase, the skill-level grows monotonically as the learning time increases. For this reason, the choice of task to be learned is an important one. We have chosen the task of shooting as the NPC becomes naturally more proficient with aiming, through the use of RL, as it encounters more states (movement and orientation of the opponent) over time. The approach is limited by the upper bound of the performance that can be achieved by the NPC and therefore the learning component has to be well defined to avoid potential local minima stagnations during the learning phase. For instance, the SEC-Bot has to adjust to the most knowledgeable policy all of the time in order to compete at the same as a Level 5 fixed-strategy opponent.

Another point to note is that our skill balancing technique is only concerned with a single task at the moment, that of weapon proficiency. In this case, tasks such as item collection, enemy avoidance and movement around the map have fixed strategy implementations. We used a small map for both the training and testing phases so that we could focus our evaluation on the shooting performance of the NPC. This is, of course, the key task in an FPS game but there is plenty of scope for learning other secondary tasks in the game and

combining them to form the balancing mechanism.

There is no guarantee that the underlying RL agent for SEC will exhibit a skillset that will suit all human players, given the different player personas and playing styles that exist. In this research, we decided to focus on the aiming efficiency of the NPC with a single weapon. We ran tests against scripted fixed-strategy opponents so that we could closely monitor the effect of the skill adjustments. We were successful in what we set out to achieve and we believe that it would be straightforward to develop tailor-made behaviours to suit each of the different types of weapon available. Having weapon-specific decision making for multiple weapons could take us a step closer to more generalised behaviour that would suit a wider variety of playing styles.

The results for SEC are very promising and show that, using a threshold mechanism and milestones from the learning timeline, we can closely match the level of five different fixed-strategy opponents (with varying degrees of proficiency) using a single instance of the SEC mechanism. We believe that this mechanism could be useful for a wide variety of game genres that produce explicit performance metrics and, in this paper, we have shown how successful it can be in the context of weapon proficiency in an FPS game.

VIII. Future Work

The following are some possible refinements that could be applied to the SEC-Bot in order to improve its skill-balancing capability.

The criteria for selecting appropriate milestones is an interesting task. Careful performance analysis could aid in the process of milestone creation to determine definitive points of performance improvement which may not follow systematic increments. For instance, we may wish to select a larger number of milestones during the earlier stages of learning and select fewer milestones as the learning begins to plateau.

A milestone/player caching system could be introduced for using the SEC against multiple opponents. Each opponent would have an ID and an associated milestone so that the SEC-Bot could switch to an appropriate milestone based on the current opponent. We designed the initial system to balance play with just a single opponent.

Finally, the SEC-Bot could be trained against different levels of human players as opposed to fixed-strategy bots. Another approach could be to deploy the SEC-Bot on a server, over the Internet, to train it against both the human and computer-controlled players that it encounters.

References

[1] Treanor, Mike and Zook, Alexander and Eladhari, Mirjam P and Togelius, Julian and Smith, Gillian and Cook, Michael and Thompson, Tommy and Magerko, Brian and Levine, John and Smith, Adam.: AI-based game design patterns. Society for the Advancement of Digital Games. (2015)

[2] Guckelsberger, Christian and Salge, Christoph and Colton, Simon.: Intrinsically motivated general companion npcs via coupled empowerment maximisation. Computational Intelligence and Games (CIG), 2016 IEEE Conference on, pp.1–8. (2016)

[3] Hunicke, Robin and Chapman, Vernell.: AI for Dynamic Difficulty Adjustment in Games. In: Challenges in Game Artificial Intelligence AAAI Workshop, vol. 2, pp. 91–96 (2004)

[4] Andrade, Gustavo and Ramalho, Geber and Santana, Hugo and Corruble, Vincent.: Extending reinforcement learning to provide dynamic game balancing. In: proceedings of the Workshop on Reasoning, Representation, and Learning in Computer Games, 19th International Joint Conference on Artificial Intelligence (IJCAI), pp. 7–12. (2005).

[5] Sutton, R. S and Barto, A. G.: Reinforcement Learning: An Introduction. Adaptive Computation and Machine Learning. MIT Press. (1998)

[6] Liandri.: Unreal Tournament 2004. http://liandri.beyondunreal.com/Unreal_Tournament_2004 [Accessed: 5-Mar-18] (2014)

[7] Gemrot, J. and Kadlec, R. and Bida, M. and Burkert, O. and Pibil, R. and Havlicek, J. and Zemcak, L. and Simlovic, J. and Vansa, R. and Stolba, M. and Plch, T. and Brom C.: Pogamut 3 Can Assist Developers in Building AI (Not Only) for Their Videogame Agents. In: Agents for Games and Simulations. Lecture Notes in Computer Science, Springer, pp. 1–15. (2009)

[8] Koster, Raph.: Theory of Fun for Game Design. O'Reilly Media, Inc. (2013)

[9] Vorderer, Peter and Hartmann, Tilo and Klimmt, Christoph.: Explaining the enjoyment of playing video games: the role of competition. Proceedings of the second international conference on Entertainment computing, pp. 1–9, Carnegie Mellon University, (2003).

[10] Abuhamdeh, Sami and Csikszentmihalyi, Mihaly and Jalal, Baland.: Enjoying the possibility of defeat: Outcome uncertainty, suspense, and intrinsic motivation. Motivation and Emotion, vol. 39(1), pp. 1–10. (2015).

[11] Rollings, Andrew and Ernest, Adams.: On Game Design. New Riders, (2003).

[12] Hunicke, Robin.: The case for dynamic difficulty adjustment in games. In: proceedings of the 2005 ACM SIGCHI International Conference on Advances in Computer Entertainment Technology, pp. 429–433. ACM, (2005).

[13] Spronck, Pieter and Sprinkhuizen-Kuyper, Ida and Postma, Eric.: Difficulty Scaling of Game AI. In: proceedings of the 5th International Conference on Intelligent Games and Simulation (GAME-ON 2004). (2004).

[14] Spronck, Pieter and Sprinkhuizen-Kuyper, Ida and Postma, Eric.: Online Adaptation of Game Opponent AI with Dynamic Scripting. International Journal of Intelligent Games and Simulation, pp. 45–53. (2004).

[15] Tan, Chin Hiong and Chen Tan, Kay and Tay, Arthur.: Dynamic Game Difficulty Scaling using Adaptive Behavior-based AI. In: IEEE Transactions on Computational Intelligence and AI in Games, pp. 289–301. (2011).

[16] Vicencio-Moreira, Rodrigo and Mandryk, Regan L. and Gutwin, Carl and Bateman, Scott.: The Effectiveness (or lack thereof) of Aim-assist Techniques in First-person Shooter Games. In: proceedings of the 32nd annual ACM conference on Human factors in Computing Systems, pp. 937–946. (2014).

[17] Vicencio-Moreira, Rodrigo and Mandryk, Regan L. and Gutwin, Carl.: Now you can compete with anyone: Balancing players of different skill levels in a First-person shooter game. In: proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems, pp. 2255–2264. ACM, (2015).

[18] Glavin, F. G. and Madden, M. G.: Adaptive Shooting for Bots in First-Person Shooter Games Using Reinforcement Learning. In: IEEE Transactions on Computational Intelligence and AI in Games. vol. 7, pp. 180–192. (2015)

[19] Glavin, F. G. and Madden, M. G.: Learning to Shoot in First-Person Shooter Games by Stabilizing Actions and Clustering Rewards for Reinforcement Learning. In: IEEE Conference on Computational Intelligence and Games. pp. 344–351. (2015)

[20] Rummery, G. and Niranjan, M.: On-line Q-learning using Connectionist Systems. Technical Report, University of Cambridge, Engineering Department. (1994)

[21] Klopf, A Harry.: Brain Function and Adaptive Systems: A Heterostatic Theory. Technical Report. Air Force Cambridge Research Labs Hanscom Air Force Base. (1972)

[22] Glavin, F. G.: Towards Inherently Adaptive First Person Shooter Agents using Reinforcement Learning. Doctoral Dissertation. (2015)

[23] Wiki Unreal.: Unreal Wiki. http://wiki.beyondunreal.com/Legacy:Bot_Mind [Accessed: 5-Mar-18] (2007)

The Evolutionary Race: Improving the Process of Evaluating Car Controllers in Racing Simulators

Mohammed Salem
Dept. of Computer Sciences
University of Mascara
Algeria
salem@univ-mascara.dz

Antonio M. Mora
Dept. of Signal Theory, Telemat. and Communic.
ETSIIT-CITIC, University of Granada
Spain
amorag@ugr.es

Juan J. Merelo
Dept. of Comp. Architect. and Technology
ETSIIT-CITIC, University of Granada
Spain
jmerelo@ugr.es

***Abstract*—Simulated car races have been used for a long time as an environment where car controlling algorithms can be tested; they are an interesting testbed for all kinds of algorithms, including metaheuristics such as evolutionary algorithms. However, the challenge in the evolutionary algorithms is to design a reliable and effective evaluation process for the individuals that eventually translates into good solutions to the car racing problem: finding a controller that is able to win in a wide range of tracks and with a good quantity of opponents. Evaluating individual car controllers involves not only the design of a proper fitness function representing how good the car controller would be in a competitive race, but also the selection of the best solution for the optimization problem being solved; this decision might not be easy when uncertainty is present in the problem environment; in this case, weather and track conditions as well as unpredictable behavior of other drivers. Creating a methodology for the automatic design of the controller of an autonomous driver for a car racing simulator such as TORCS is an optimization problem which offers all these challenges. Thus, in this paper we describe an analysis and some proposals to improve the evaluation of optimized fuzzy drivers for TORCS over previous attempts to do so. It builds on preliminary results obtained in previous papers as a baseline and aims to obtain a more competitive autonomous driver via redesign of the fitness evaluation procedure; to this end, two different fitness functions are studied in several experiments, along with a novel race-based approach for the selection of the best individual in the evolution.**

***Index Terms*—simulated car racing, TORCS, fuzzy controllers, autonomous drivers, genetic algorithms, optimization, evaluation**

I. Introduction

Autonomous driving is a problem that shows up in many environments, including of course self-driving cars, but also in drones, ships, trains and underwater vehicles. In general, there will be a series of sensor inputs that include real speed, obstacles and other vehicles, and based on those sensors, the bot will have to take a decision on speed and steering that is optimal in several different senses [13]. Testing different autonomous driving methodologies in real life is usually reserved to just a few big players, and methodologies as well as algorithms are usually tested in simulated environments; these simulated environments, at the same time, offer the incentive of competition among your system and others. In this paper, we will be using the Open Racing Car Simulator (TORCS) [28], a very realistic racing simulator which offers a great testbed for the implementation and evaluation of autonomous drivers. It has been used several times for the celebration of Artificial Intelligence (AI) competitions, where the aim is to create the best autonomous driver for racing [16]–[18]. Besides being able to test your car against other cars that have been published, it can be used as a standalone environment to optimize driving in a solo race.

Evolutionary Algorithms (EAs) [1] have been frequently applied as a general-purpose optimization method in this area, generally combined with behavioural engines that rule different parts of the car [5], [11], [23], [24]. These driving engines have included lately fuzzy controllers [8], [15], [22]. These controllers use fuzzy Logic [6], a technique that is quite suitable for defining this kind of autonomous agents, since they are in part inspired by the human reasoning when driving. A fuzzy controller works with linguistic variables, and will for instance turn *slightly* to the right when the next curve is *close*, but these controllers have to be designed to map properly inputs to desired outputs in particular situations. Previously, the authors presented an approach combining two specialized fuzzy controllers, designed by hand, that were able to decide the car's proper steering angle and desired speed at every single point (or tick) during a race [26]. This driver was later improved [27] optimizing the parameters of their membership functions by means of a Genetic Algorithm [7]; this automated design improved manual one obtaining several controllers that were able to beat the initial hand-designed controller in a race, as well as other published controllers.

This proved that evolutionary algorithms were able to get the fuzzy controller parameters better than a hand-made design, but at the same time revealed several challenges. In general, evolutionary algorithms optimize the fitness function that is used; evolved fuzzy controllers (hereafter FCs) will be eventually as good as the fitness function allows. But in this particular case we cannot use as fitness function the position obtained by the FC in every possible race on every possible track with every possible opponent, so we have to settle for a *surrogate* of the fitness in a very limited environment. First we opted for eliminating opponents and making evaluations in solo races; then we chose a particular track that combined straight segments as well as some curves and did not take too long to run, and eventually we had to decide what factors

978-1-5386-4359-4/18/$31.00 ©2018 IEEE

related to speed, damage and lap time were going to be effectively included in the final fitness function.

Results were encouraging, but it is still a surrogate model. As such a model, we need to decide on the best track to perform *training* and also the solo race measures with the biggest impact in the eventual racing performance. This is why in this paper we have combined into the fitness function only those terms related with speed during race (to maximize) and damage (to minimize) - the most important factors - in two different approaches. Besides, the fitness evaluation process has a certain amount of uncertainty because damages and some track conditions can randomly vary in different evaluations. This is why instead of selecting directly the driver as we did before, we will be using an actual race among the best drivers to select the best one.

The main objective of the three new techniques introduced in this paper, namely, heuristic track selection, fitness function design and *winner* selection have as the main objective to create a better model of the racing environment. Its results will be commented in the corresponding section.

The rest of the paper is organized as follows. Next we present the state of the art, to be followed by a description of the TORCS simulator, the fuzzy controllers and the Genetic Algorithm implemented in Section III. After it, the experiments conducted and the obtained results are described in Section IV. Finally, conclusions and future lines of work will be presented in section V.

II. State of the Art

TORCS has become one of the main environments for research on AI since its launch in 2007 [28]. It offers different kinds of problems to solve, such as the definition of the optimal parameters for the cars (before the race) [12], and the main one, the design of competitive autonomous drivers aiming to win races against other cars, mainly presented in the context on the Simulated Car Racing Competition [16], [17].

Evolutionary algorithms have targeted TORCS almost since its publication, for instance, for determining the optimal trajectory of a lap in a known circuit [25], but this approach suffers from the problem that the obtained trajectory in the evolving process strongly depends on the initial state of the car. In the same context, the authors in [19] tried to design a novel approach to compute the optimal racing line without any human intervention, using a GA to find the best trade-off between the minimization of two conflicting objectives: the length and the curvature of the racing line.

However, definitely, the most prolific area of application of EAs inside TORCS has been the optimization of autonomous controllers for car driving, i.e. conducting a meta-optimization process. Thus, EAs have been applied to 'refine' the parameters which define the driver's behavior [2], [11], or to improve the structure/architecture of the models [11], [14], working offline, or online (during the game) [3], [29]. Our approach is focused in this line, proposing the application of an off-line genetic algorithm for the improvement of the parameters which determine the behavior of a controller for TORCS. We have focused on a Fuzzy-based model, as it is one of the best options for modeling human-like decisions and actions, as others authors have also used this kind of technique in the literature with good results [23]. For instance, in [8], a fuzzy rule-based car controller for a Car Racing Competition was built and tuned with co-evolutionary genetic algorithms. Two fuzzy sub-controllers were designed (acceleration and turning angle). But this approach was applied to a simpler simulator than TORCS which is more realistic and time-constrained.

Pérez *et al.* introduced an evolutionary fuzzy approach for TORCS in [15], where they applied EAs for improving fuzzy models to infer the acceleration and turning angle. However, the models were not so specialized as the proposed here, since their controller did not compute the target speed, which is a key factor for a competitive controller.

Onieva *et al.* [22] presented a parametrized modular architecture with a fuzzy system and a GA, aiming to reproduce the behavior of a human driver when controlling the steering wheel. In this line, we presented an improved approach [26], which evolved a fuzzy-based driver considering the target speed in addition to the steer (two fuzzy sub-controllers). This controller was also enhanced in a further work [27] optimizing the parameters of the membership functions by means of a real-coded GA, obtaining a noticeably improved performance.

In this paper we focus on the improvement of the evolved controllers by means of a redefinition of the fitness functions, looking for a parameter-less approach (no weights in the terms) [9], which will be also more focused on the real objectives for a driver during a race, rather than the overall target. In addition, a 'fairer' selection process of the best individual has been implemented, aiming to focus on real good drivers in races, instead of choosing the one with the highest fitness value, which could have a poorer performance due to the uncertainty/noise present on the problem.

III. Experimental Setup

The Open Racing Car Simulator (TORCS) [28] is an open source, modern, multi-player, modular and portable racing simulator that allows users to compete against other computer-controlled opponents. Its high degree of modularity and portability, together with the realistic and real-time driving simulation, make it an ideal testbed for artificial intelligence research, as we have shown in the previous section.

Every car in TORCS includes a large set of sensors, whose values the car can use during a race, such as distances to track borders, to rivals, current fuel, current gear, position in the race, speed, or damage, among others. The sensor values can be considered by any TORCS autonomous driver, or *controller*, to manage the car using actuators: the steering wheel, the accelerator, the brake pedal and the gearbox. The designed controller is based on the model of sensors and actuators of the simulated car racing competition [18].

A. Fuzzy sub-controllers

The controller proposed initially [26] has the same modular architecture as the simple TORCS driver; however, the target

speed and steering angle are computed by means of two modular and specialized fuzzy sub-controllers, which consider five position sensors. This is the controller which will be improved by means of a GA in this work.

The *fuzzy target speed sub-controller* aims to estimate the optimal target speed of the car, both in straight parts and curves of the track, taking into account two criteria: move as fast as possible and be safe. This estimation is based on two general cases: if the car is in a straight line, the target speed will take a maximum value (*maxSpeed* km/h). However, if it is close to a curve, the controller will decrease the current speed to a value included in the interval *[minSpeed, maxSpeed]* km/h.

This fuzzy controller has an output, the speed, and three input values:

- Front = Track_9: front distance to the track border (angle 0°).
- M5 = max (Track_8, Track_10): max distance to the track border in an angle of +5°and -5°with respect to Front.
- M10 = max (Track_7, Track_11): max distance to track border in an angle of +10°and -10°.

It is a Mamdani-based fuzzy system [10] with three trapezoidal Membership Functions (MF) for every input variable. In [27] the different sets of parameters which define the membership functions were improved using a Genetic Algorithm to obtain the best results.

Moreover, the controller is based in a set of fuzzy rules, designed to maximize the car speed depending on the distance to the track border. These rules can be consulted in [26]. The second is the *fuzzy steering sub-controller*, which aims to define the steer angle estimating and determining the target position of the car. The structure of this sub-controller is similar to the speed one, but it has the steering as output. Thus, the set of sensors considered is the same as in the speed case.

Then, as general rules: if the car is in a straight line, it will set as target position half width of the race track (central position of the lane). Whereas, if the car is near a right/left curve, it will approach the path leading to the right/left, with a space between the car and the border of the track to avoid the loss of control.

In order to detect the curves, the controller focuses on the sensor values (M10, M5, and Front). So, if the value on Front sensor is the longest, there is a straight road; whereas if the values of M5 and M10 with positive angles (+5 and +10) are the longest, there is right curve; and the other way round.

It uses a base of rules which has been defined trying to model the behavior of a human driver [26].

B. Genetic Algorithm

The proposed optimization approach aims to find the optimal parameters of the membership functions of the two sub-controllers previously introduced. The followed process is depicted in Figure III-B, in which, as it can be seen, the GA uses TORCS for the evaluation of every individual during the evolutionary process.

The GA starts by creating the initial population with random values for the parameters in the defined range $[0, 100]$. The

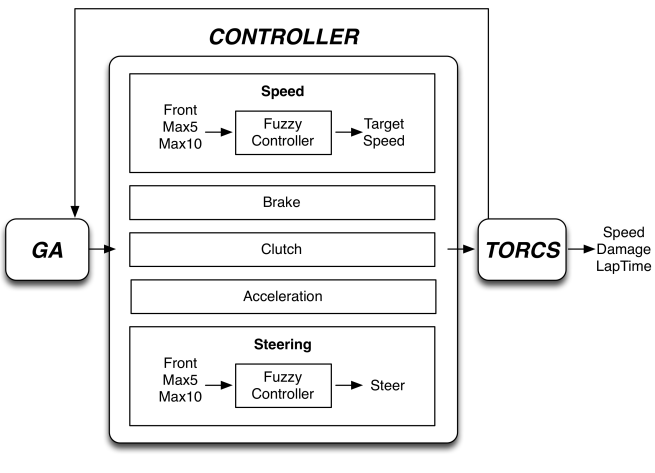


Fig. 1. Optimization of a fuzzy controller flowchart. The evaluation of an individual is performed by: putting the parameter values on the two sub-controllers, launching a race in TORCS with this configuration, obtaining the resulting values of Damage, Top Speed and mean Lap Time and using these values for the computation of the fitness of the individual.

fitness of each candidate solution is computed by injecting its gene values to the parameters of the membership functions of the two fuzzy sub-controllers. The defined autonomous controller is used to drive a car in a 20 laps race in a circuit without opponents, and the results (Maximum, Minimum and Average speed, Damage) are used to compute the fitness value.

As previously stated, the designed fuzzy controllers have trapezoidal membership functions given by Equation 1. In such a controller, fuzzy rules are applied to linguistic terms. These terms, which qualify a linguistic variable, are defined through membership functions, which, in turn, depend on a set of parameters that 'describes' their shape (and operation). Using a GA we will optimize the parameters of the membership functions that constitute the fuzzy partition of the linguistic variable [30]. The input linguistic variables in our problem, *Front, Max5* and *Max10*, are represented by three trapezoidal membership functions.

A trapezoidal membership function in a finite universe of discourse *[a, b]* can be defined by:

$$\mu_A(x) = \begin{cases} \frac{x-x_1}{x_2-x_1}, & x_1 \leq x \leq x_2 \\ 1, & x_2 \leq x \leq x_3 \\ \frac{x_4-x}{x_4-x_3}, & x_3 \leq x \leq x_4 \\ 0, & else \end{cases} \quad (1)$$

with:

$$x_1 \leq x_2 \leq x_3 \leq x_4 \quad (2)$$

This MF function is defined by four parameters x_1, x_2, x_3 and x_4 taking their values in the interval *[a, b]* (Figure 2).

And a fuzzy partition with *n* trapezoidal membership functions is defined by *2n* variables ($a = x_1, x_2, .., x_{2n} = b$)(Equation 4). In this case, the representation is given by Figure 3. With:

$$a = x_1 \leq x_2 \leq ... \leq x_{2n-1} \leq x_{2n} = b \quad (3)$$

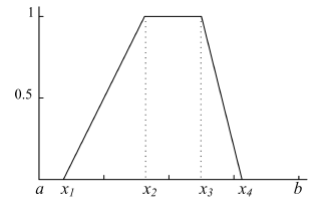


Fig. 2. Trapezoidal MFs

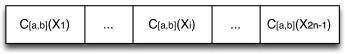


Fig. 3. Trapezoidal-shaped MFs coding

$$\mu_{A1}(x) = \begin{cases} 1, & x_1 \leq x \leq x_2 \\ \frac{x_3 - x}{x_3 - x_2}, & x_2 \leq x \leq x_3 \\ 0, & x > x_3 \end{cases}$$

$$\mu_{Ai}(x) = \begin{cases} 0, & x \leq x_{2i-2} \\ \frac{x - x_{2i-2}}{x_{2i-1} - x_{2i-2}}, & x_{2i-2} \leq x \leq x_{2i-1}, n = 2, ..., i-1 \\ 1, & x_{2i-1} \leq x \leq x_{2i} \\ \frac{x_{2i+1} - x}{x_{2i+1} - x_{2i}}, & x_{2i} \leq x \leq x_{2i+1} \\ 0, & x > x_{2i+1} \end{cases}$$

$$\mu_{An}(x) = \begin{cases} 0, & x \leq x_{2n-2} \\ \frac{x - x_{2n-2}}{x_{2n-1} - x_{2n-2}}, & x_{2n-2} \leq x \leq x_{2n-1} \\ 1, & x > x_{2n-1} \end{cases} \quad (4)$$

When the number of parameters is reduced and their ranges of variations are well defined, a GA with a binary coding is largely sufficient to find their optimal values. On the other hand, if the number of parameters becomes important, and their variation interval is not well known, the real coding is the most appropriate [4]. Since our work requires some precision and the variation interval of each parameter is not well known, we have considered a real coding implementation in a vector that includes all variables to optimize.

In that GA, every individual is a vector of 18 values/parameters, 6 per variable. Figure 4 illustrates the structure of the chromosome.

The initialization of the chromosomes (first population) is performed assigning random values inside a range of variation [7], in order to start from feasible values [26]. Tournament based selection has been used to elect chromosomes as parents for genetic operators, while simple arithmetic two point crossover [31] and non uniform mutation [21] have been chosen, as two of the most contrasted methods in the literature.

The objective of the car controller is to win as many races as possible. However, we have to optimize the most general case by carrying out solo *training races*, which will be less sensitive to the presence of noise/uncertainty due to the participation of other controllers.

In previous approaches we focused on minimizing the damage of the car ($damage$) and the lap time $LapTime$, while maximizing $TopSpeed$. However in this study, we have turned our focus to a more 'human-like approach', i.e. trying to drive as fast as possible in every single part of the track while avoiding damage. Thus, we have considered:

- $MinSpeed$: aiming to increase the good driving in the difficult zones of the tracks (e.g. curves).
- $MaxSpeed$: centered on the enhancement of the controller in easy or straight parts.
- $AVG(Speed)$: which shows the combination of the overall behavior in the whole track.
- $Damage$: aiming to create 'safe' controllers, as it is mandatory being able to finish the race.

We have combined these terms into two possible fitness functions:

GFC-MMS:

$$f_1 = \frac{(MinSpeed * MaxSpeed)}{Damage+1} \quad (5)$$

GFC-AVS:

$$f_2 = \frac{AVG(Speed)}{Damage+1} \quad (6)$$

As it can be seen, the aim is to maximize the minimum and maximum velocities while minimizing the damage in the first fitness. In the second one, we try to maximize the average speed on the complete circuit.

To evaluate every candidate controller during the evolutionary process, we will make each of them compete in a 20 laps practice race in a medium difficulty circuit without rivals. As stated, we have omitted the presence of opponents in order to avoid including additional uncertainty sources to the optimization process. In order to obtain general-behaviour controllers, the selected track for this process will be one with a combination of curves and straight parts.

Once this practice is finished, the obtained output values $Damage$, $MinSpeed$, $MaxSpeed$ and $AVG(Speed)$ are collected to compute the corresponding fitness value.

IV. Experiments and results

After analyzing most of the available tracks, we have selected for these experiments the **Alpine 2** circuit. It is a quite complex one, with multiple turns, but also straight parts (See Figure 5).

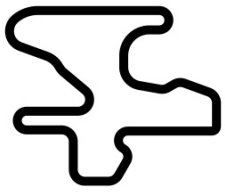


Fig. 5. Alpine 2 Track: Slow mountain road. Length: 3773.57m, Width: 10m

We have used the driving car *car1-tbr1* for our controllers, since according to previous experiments [26], this is a fair

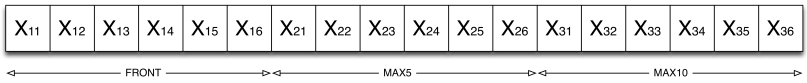


Fig. 4. Chromosome description

choice due to its moderate performance. This will lead our controller to be prepared to drive in the most usual conditions.

We have evaluated the Genetic Fuzzy Controller (GFC) with the two proposed fitness functions: GFC-MMS (Equation 5) and GFC-AVS (Equation 6), comparing them for racing performance. We have run the algorithm with a population size of 50 individuals. The rest of parameters are: Generations=50, Crossover rate=0.85, Mutation rate=0.1, and 10 different runs per configuration.

The two genetic optimization processes have been run independently. However, as a difference to previous works in which we just selected as the best the individual with the highest fitness value in the last generation, in this study we have aimed to implement a better methodology, which we expect will yield a more competitive controller.

To this end, once the evolutionary process is finished and for every approach, the best four individuals have competed together in 5 races (of 5 laps) in the **Alpine 2** track (used in the optimization) and 5 races (of 5 laps) in **E-Track 5** track (a new one for them). In order to enhance the selection of the best, another two controllers are picked randomly from the default TORCS bots to participate in the race. We have implemented a *score-based competition*, which is based in Formula 1 schema, so the obtained scores depend on the car position: 1 - 25 points, 2 - 18, 3 - 15, 4 - 12, 5 - 10, 6 - 8, 7 - 6, 8 - 4, 9 - 2, 10 - 1. Moreover, in order to consider the fitness terms, we have defined an *extra score*, so the controller which gets the fastest time or the minimum damage in each lap is given 5 extra points.

The results of these runs are shown in Table I. "Best laps" and "Min damage" are the scores obtained by every controller in every race when getting the best lap time or/and the minimum damage of all the contenders. The symbol '-' means the TORCS bot do not participate in the race.

According to the table, the first individual of GFC-MMS and the second of GFC-AVS have won the same number of races but $GFC-AVS_2$ has achieved better scores on the races it did not win. We have to remark that the TORCS bots results are not counted since they only serve to diversify the selection and they do not participate in all the races. This selection allowed us to choose the best individual on several races and the most stable, and thus, avoid the classical selection by tournament where the winner of a single confrontation is chosen.

It could be also pointed out that the genetic-based fuzzy controllers have get all the points of the minimum damage even when they do not win the race, this fact strongly justifies the use of damage in the fitness functions, which is a key factor to consider in real races.

The controllers obtained using the first fitness have also collected the points of the best laps in five out of the ten races. We have to bear in mind that the best lap is the result of a minimum damage and a high MaxSpeed, which are both optimized by the first function. In the same line, the second fitness tries to maximize the average speed, not necessarily the MaxSpeed.

For the following experiment, we have chosen the two - one per fitness function - best genetic-based fuzzy controllers obtained in the previous tests, which are named $GFC-MMS_1$ and $GFC-AVS_2$. They have been evaluated in a set of practice races against some selected opponents. In addition, the two best evolutionary controllers of our previous paper [27], $EVO1$ and $EVO2$, have been also included in the competition.

This evaluation is a type of mini-championship, which also considers Formula 1 scores. It consists of 10 races, each one for 20 laps, and with a total of 10 participants per race: the best two $GFC-MMS_1$ and $GFC-AVG_2$, $EVO1$, $EVO2$, and also 6 bots from TORCS are used to test our controllers. The first 5 races are conducted in **Alpine 2** track (trained one); and the other 5 races took place in **E-Track 5** track (not trained for the new controllers, but used in the evolution of our previous ones). The the starting grid (initial positions of cars) on these races was set randomly. It is important to note that in these races there are not extra points.

The results are shown in Table II. This table shows how one of the fuzzy controllers that are evolved using the new selection and evaluation functions $\mathsf{GFC\text{-}MMS}_2$ yields the best results, obtaining very good overall rankings in the races. Inferno1 controller also obtained very good results, getting 3 times the best lap score, but it happened due to it used the fastest car. We can also see that our two genetic based fuzzy controllers have only won: one race $GFC-MMS_1$ and two races $GFC-AVS_2$, while $berniw2$ and $inferno1$ bots have won three races each one. To complement the visualization of results, we have plotted the overall number of points of the championship in Figure 6. Figure 7 represents the number of races where each controller has obtained the minimum Damage and the highest Maximum Speed.

As it can be seen in the table and figures, even if they are not able to win, our new genetic controllers have finished in the first four positions in the races, which helped them to win the championship by score. This result is due to the trade-off between the damage and the average speed achieved by the fitness function (mainly the second one). This property has given our controllers an advantage in minimizing damage and looking for top speed in the straight sections of the track while it tries to find the best trajectory in turns to avoid hitting the

TABLE I
OBTAINED SCORES ON THE RACE-BASED SELECTION FOR THE TWO APPROACHES IN TWO DIFFERENT TRACKS.

Driver	5 races in Alpine 2 track (5 laps each)							5 races in E-Track 5 (5 laps each)							
	R1	R2	R3	R4	R5	Best laps	Min damage	R1	R2	R3	R4	R5	Best laps	Min damage	Total
$GFC-MMS_1$	**25**	18	8	15	12	15	10	12	**25**	18	**25**	18	10	10	**221**
$GFC-MMS_2$	12	**25**	15	12	15	0	5	8	15	15	4	10	0	0	136
$GFC-MMS_3$	6	6	10	10	8	0	0	15	10	10	18	6	0	5	104
$GFC-MMS_4$	2	8	4	4	6	0	0	10	1	2	1	2	0	0	40
$GFC-AVS_1$	1	4	6	2	4	0	0	4	2	12	10	4	0	0	49
$GFC-AVS_2$	15	10	18	**25**	18	5	10	**25**	18	**25**	15	15	5	10	206
$GFC-AVS_3$	10	2	1	6	2	0	0	2	6	4	6	1	0	0	41
$GFC-AVS_4$	4	1	2	1	1	0	0	1	4	8	2	8	0	0	32
$bt1$	-	-	-	8	-	0	0	-	8	6	8	-	0	0	-
$inferno1$	18	-	12	-	-	0	0	18	12	-	-	-	0	0	-
$berniw2$	8	15	-	18	-	0	0	-	-	-	12	12	5	0	-
$berniw3$	-	12	-	-	**25**	5	0	4	-	-	-	**25**	5	0	-
$damned1$	-	-	**25**	-	10	0	0	-	-	8	-	-	0	0	-

TABLE II
RESULTS OF THE MINI-CHAMPIONSHIP WITH 10 DRIVERS AND 10 RACES IN TWO DIFFERENT TRACKS. `tita`, `berniw` AND `inferno` ARE EXAMPLE CONTROLLERS INCLUDED WITH THE TORCS SIMULATOR [28]

Driver	Races in **Alpine 2** track (20 laps each)						Races in **E-Track 5** track (20 laps each)						Total Score
	R1	R2	R3	R4	R5	Track Score	R6	R7	R8	R9	R10	Track Score	
$GFC-MMS_1$	25	10	18	12	10	75	18	12	15	15	12	72	147
$GFC-AVS_2$	15	18	25	15	15	**88**	25	18	18	12	18	91	**179**
$tita1$	4	2	1	2	2	11	4	2	1	4	6	17	28
$tita2$	2	1	2	1	1	7	1	1	2	1	2	9	16
$inferno1$	12	15	12	18	18	75	12	15	25	25	15	**92**	167
$inferno2$	10	12	4	10	25	61	10	10	4	2	8	34	95
$berniw1$	18	25	15	8	6	72	8	8	6	10	10	42	114
$berniw2$	8	8	10	25	12	63	15	25	10	8	25	83	146
$EVO1$	6	6	8	4	8	32	2	6	12	8	4	32	64
$EVO2$	1	4	6	6	4	21	6	4	8	6	2	26	47

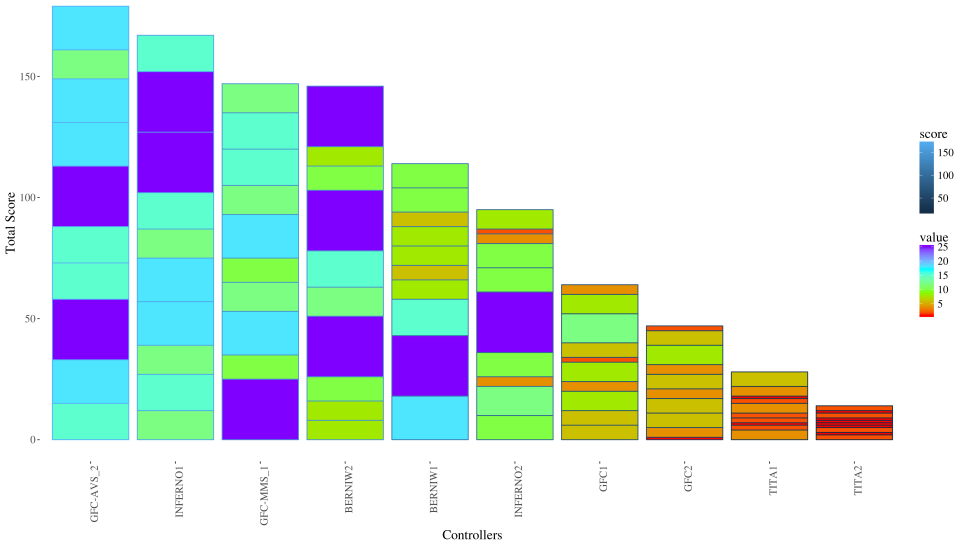


Fig. 6. Total and partial score for all controllers in the *championship*. The two controllers introduced in this paper use the `GFC` prefix.

track edges or get out of it.

Also, as we can see in Figure 7 $GFC-MMS_1$ is better at doing fast laps, and at the same time it sustains less damage. Peak speed is not enough to eventually win a race. Figure 6 shows how $GFC-AVS_2$ achieves the best score in the *championship*. It should be noted that this victory is mainly achieved thanks to the victories in `E-Track 5`. Although it is always in a good position in this track, $GFC-MMS_1$ fails to win a single race, being overcome by the other tested controller as well as $inferno1$.

Finally, Table III presents a comparison between the genetic based fuzzy controllers presented in this paper, $GFC-MMS_1$ and $GFC-AVS_2$ with those obtained in [27], $EVO1$ and $EVO2$. The results are the average values of $damage$, $MaxSpeed$ and $Speed$ of 10 races in the **Alpine 2** and **E-Track 5** tracks.

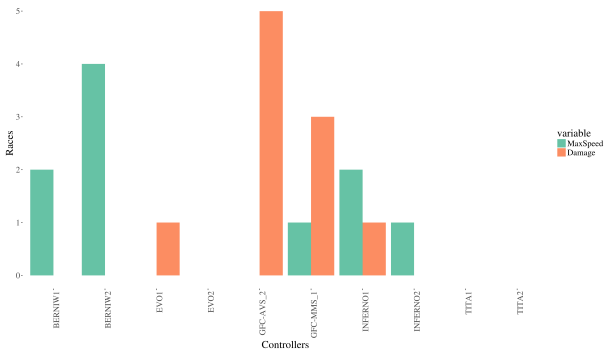


Fig. 7. Bar chart with the number of races where controlled cars obtain the minimum damage and where they obtain the highest MaxSpeed in a lap.

TABLE III
AVERAGE DAMAGE AND SPEED RESULTS OF GFC IN 5 RACES IN **ALPINE 2** AND 5 RACES IN **E-TRACK 5** TRACKS

Alpine 2				
	$GFC-MMS_1$	$GFC-AVS_2$	$EVO1$	$EVO2$
Average Speed (km/h)	188.66	194.39	154.02	150.98
Max Speed (km/h)	229.77	218.91	213.42	215.00
Damage	122.43	134.85	124.02	131.41
E-Track 5				
	$GFC-MMS_1$	$GFC-AVS_2$	$EVO1$	$EVO2$
Average Speed (km/h)	166.89	169.33	147.92	145.78
Max Speed (km/h)	252.72	241.29	236.33	231.90
Damage	22.89	18.44	23.34	15.26

The average values obtained for the maximum speed and the damage show a clear improvement for the two new controllers, with regard to previous ones, given by the novel fitness functions proposed here. Although the damage values of $GFC-MMS_1$ and $GFC-AVS_2$ remains similar to the ones obtained by 'EVOs', the improvement in the average and maximum speed reached during the races is clear. This fact is even more remarkable in the case of $GFC-MMS_1$, which was designed with a fitness function in which the speed had a higher relevance. On the other hand, we can notice the considerable increase in the average speed of controller $GFC-AVS_2$ in both tracks and especially in Alpine 2, which was the one used during the evolution process. This increase translates into a decrease in race time in each segment of the track and therefore an overall better performance. According to the results in the table we could argue that the evaluation functions are working properly. And moreover, that the new approaches are working much better than previous ones, even in the circuit in which the 'EVOs' were trained.

V. CONCLUSIONS AND FUTURE WORK

In this work, we have presented new ways of evaluating controllers for the TORCS game; these new methods work within an evolutionary algorithm that optimizes a set of fuzzy controllers for TORCS cars [27]. It combines two sub-controllers, one to calculate the target speed and the other for the direction, that is, for driving the steering wheel.

After initial tests in previous works [26], [27], that showed the promise of using evolutionary algorithms with two different fitness functions, one considering the average lap time and the car damage and another adding the top speed reached, we discovered the importance of the speed, using different combinations. *A priori*, we could think that races are won by being very fast when you can be, and trying not to get too slow in tortuous segments. This heuristic was what drove us to design the first evaluation function, $GFC-MMS$. However, it could be thought that sustaining a high average speed will be the most important factor; this was the heuristic used for $GFC-AVS$. The choice of the single track used for training reflected these facts: tracks usually combine "fast" and "slow" segments, and the car should be able to perform well on both.

Besides, the uncertainty of the evaluation is taken into account when selecting the *best* controller. Since there is a random element in evaluation, instead of just picking the individual with the best fitness in the last generation, we select the best ones, make them complete against each other and other rivals in a set of real races in several tracks; then select the one with the best results overall score. Introducing this element of reality also makes the choice of controller for comparison a more robust choice than previously.

The yielded results are very promising since the optimized controllers (one per fitness function) were ranked among the first ones in different evaluation races with rivals, with the minimum of damage and a very good average and maximum speed.

In the comparison with the previous evolved fuzzy controllers (from [27]), the improvement can be clearly seen in the results. The new controllers are able to drive much faster than them, even in a track where the latter were trained, and moreover they manage to sustain a low damage. They do so even if, compared with [26], [27], they are not using damage as a criterium for optimization. High damage might make a car abandon a race, but apparently a certain amount of damage should be expected if you want to win the race.

At this point, it would be difficult to say which of the three factors introduced in this paper, namely, new evaluation functions, race track for evaluation and post-optimization selection of winner, is the one that contributes the most to the good results. Clearly all three of them have an impact, but, taking into account the big difference among results in the post-optimization selection of controllers in the last generation, probably this step, which deals with uncertainty in the evaluation of the controller, might have had the biggest impact. This way of dealing with uncertainty, which is usually a problem in game bot optimization [20], might be extended in different ways, for instance as a selection method for all generations.

This is one of the future lines of work, combined with a study of the influence of all factors in bot performance. This will have to include tracks, fitness functions and post-optimization selection. This will have to be combined with

the improvement of the controllers considering a 'deeper' and 'wider' evaluation (more laps and more tracks). There is also a similar research line to address, which is the optimization of the outputs in the fuzzy system, which has been set manually in all our approaches.

With regard to the applied GA, it could be improved in different ways, for instance, reducing its computation time by means of the parallelization of the evaluation phase. Also, a multi-objective approach could be implemented, in which the main objectives to address by the controller could be optimized at once. Moreover, we could also try to generate, optimize and tune automatically the rule base of the fuzzy controller by means of a Genetic Programming algorithm.

ACKNOWLEDGEMENTS

This work has been supported in part by: Ministerio español de Economía y Competitividad under project TIN2014-56494-C4-3-P (UGR-EPHEMECH), TIN2017-85727-C4-2-P (UGR-DeepBio) and TEC2015-68752 (also funded by FEDER).

REFERENCES

[1] T. Bäck, *Evolutionary algorithms in theory and practice*. Oxford University Press, 1996.

[2] M. V. Butz and T. D. Lönneker, "Optimized sensory-motor couplings plus strategy extensions for the TORCS car racing challenge," in *Proceedings of the 2009 IEEE Symposium on Computational Intelligence and Games, CIG 2009, Milano, Italy, 7-10 September, 2009*, 2009, pp. 317–324.

[3] L. Cardamone, D. Loiacono, and P. L. Lanzi, "On-line neuroevolution applied to the open racing car simulator," in *Proceedings of the Eleventh Conference on Congress on Evolutionary Computation*, ser. CEC'09. Piscataway, NJ, USA: IEEE Press, 2009, pp. 2622–2629.

[4] S. M. M. Elsayed, R. Sarker, and D. L. Essam, "A genetic algorithm for solving the CEC2013 competition problems on real-parameter optimization," in *IEEE Congress on Evolutionary Computation, CEC 2013*, Cancun, Mexico, 21-23 June 2013 2013, pp. 356–360.

[5] D. Floreano, T. Kato, D. Marocco, and E. Sauser, "Coevolution of active vision and feature selection," *Biological Cybernetics*, vol. 90, pp. 218–228, 2004.

[6] S. Godil, M. Shamim, S. Enam, and U. Qidwai, "Fuzzy logic: A 'simple' solution for complexities in neurosciences?" *Surg Neurol Int.*, pp. 2 – 24, 2011. [Online]. Available: https://doi.org/10.4103/2152-7806.77177

[7] D. E. Goldberg, *Genetic Algorithms in search, optimization and machine learning*. Addison Wesley, 1989.

[8] S. Guadarrama and R. Vazquez, "Tuning a fuzzy racing car by coevolution," in *Genetic and Evolving Systems, GEFS 2008*, March 2008. [Online]. Available: https://doi.org/10.1109/GEFS.2008.4484568

[9] G. R. Harik and F. G. Lobo, "A parameter-less genetic algorithm," in *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation - Volume 1*, ser. GECCO'99. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999, pp. 258–265.

[10] I. Iancu, *A Mamdani Type Fuzzy Logic Controller*. InTech, 2012, pp. 325–352.

[11] T. S. Kim, J. C. Na, and K. J. Kim, "Optimization of an autonomous car controller using a self-adaptive evolutionary strategy," *International Journal of Advanced Robotic Systems*, vol. 9, no. 3, p. 73, 2012. [Online]. Available: https://doi.org/10.5772/50848

[12] M. Köle, A. S. Etaner-Uyar, B. Kiraz, and E. Özcan, "Heuristics for car setup optimisation in torcs," in *12th UK Workshop on Computational Intelligence (UKCI)*, Sept 2012, pp. 1–8. [Online]. Available: https://doi.org/10.1109/UKCI.2012.6335749

[13] S. Kolski, D. Ferguson, C. Stacniss, and R. Siegwart, "Autonomous driving in dynamic environments," in *In Proceedings of the Workshop on Safe Navigation in Open and Dynamic Environments at the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Beijing, China, 2006.

[14] J. Koutnik, G. Cuccu, J. Schmidhuber, and F. Gómez, "Evolving large scale neural networks for vision based torcs," in *Foundations of Digital Games*. Dipartimento tecnologie innovative Istituto Dalle Molle di studi sull'intelligenza artificiale: J.Koutnik, March 2013. [Online]. Available: http://repository.supsi.ch/id/eprint/4548

[15] D. P. Liébana, G. Recio, Y. Sáez, and P. Isasi, "Evolving a fuzzy controller for a car racing competition," in *Proceedings of the 2009 IEEE Symposium on Computational Intelligence and Games, CIG 2009, Milano, Italy, 7-10 September, 2009*, 2009, pp. 263–270. [Online]. Available: https://doi.org/10.1109/CIG.2009.5286467

[16] D. Loiacono, P.-L. Lanzi, J. Togelius, E. Onieva, D. A. Pelta, M. . Butz, T. D. Lonneker, L. Cardamone, D. Perez, Y. Saez, M. Preuss, and J. Quadflieg, "The 2009 simulated car racing championship," *IEEE Trans. Comput. Intell. AI Games*, vol. 2(2), pp. 131–147, 2010.

[17] D. Loiacono, J. Togelius, P. L. Lanzi, L. Kinnaird-Heether, S. M. Lucas, M. Simmerson, D. Perez, R. G. Reynolds, and Y. Saez, "The wcci 2008 simulated car racing competition," in *2008 IEEE Symposium On Computational Intelligence and Games*, Dec 2008, pp. 119–126.

[18] D. Loiacono, L. Cardamone, and P. L. Lanzi, "Simulated car racing championship: Competition software manual," *CoRR*, vol. abs/1304.1672, 2013. [Online]. Available: http://arxiv.org/abs/1304.1672

[19] D. Loiacono, P. L. Lanzi, and A. P. Bardelli, "Searching for the optimal racing line using genetic algorithms," in *Proceedings of the Symposium on Computational Intelligence and Games (CIG), 2010 IEEE*. Copenhagen, Denmark: IEEE Press, 2010.

[20] J. Merelo, Z. Chelly, A. Mora, A. Fernández-Ares, A. I. Esparcia-Alcázar, C. Cotta, P. de las Cuevas, and N. Rico, "A statistical approach to dealing with noisy fitness in evolutionary algorithms," in *Computational Intelligence*. Springer, 2016, pp. 79–95.

[21] A. Neubauer, "A theoretical analysis of the non-uniform mutation operator for the modified genetic algorithm," in *Proceedings of the IEEE International Conference on Evolutionary Computation*, 1997. [Online]. Available: https://doi.org/10.1109/ICEC.1997.592275

[22] E. Onieva, J. Alonso, J. Pérez, and V. Milanés, "Autonomous car fuzzy control modeled by iterative genetic algorithms," in *Fuzzy Systems*, 2009, pp. 1615 – 1620.

[23] E. Onieva, D. Pelta, J. Godoy, V. Milanés, and J. Rastelli, "An evolutionary tuned driving system for virtual car racing games: The autopia driver," *International Journal of Intelligent Systems*, vol. 27, pp. 217–241, 2012.

[24] E. Onieva, D. A. Pelta, J. Alonso, V. Milanés, and J. Pérez, "A modular parametric architecture for the torcs racing engine," in *Proceedings of the 5th IEEE Symposium on Computational Intelligence and Games (CIG'09)*. Piscataway, NJ, USA: IEEE Press, 2009, pp. 256–262.

[25] Y. Saez, D. Perez, O. Sanjuan, and P. Isasi, "Driving cars by means of genetic algorithms," in *Parallel Problem Solving from Nature, PPSN 2008, Lecture Notes in Computer Science, vol 5199*. Springer, Cham, 2008.

[26] M. Salem, A. M. Mora, J. J. Merelo, and P. Garca-Snchez, "Driving in TORCS using modular fuzzy controllers," in *Applications of Evolutionary Computation. EvoApplications 2017, Lecture Notes in Computer Science, vol 10199*, S. K. Squillero G., Ed. Springer, Cham, 2017, pp. 361–376.

[27] ——, "Evolving a TORCS modular fuzzy driver using genetic algorithms," in *Applications of Evolutionary Computation. EvoApplications 2018, LNCS*, K. S. et al., Ed. Springer, 2018, p. To appear.

[28] Sourceforge, "Web torcs," Web, Nov. 2016, http://torcs.sourceforge.net/.

[29] C. H. Tan, J. H. Ang, K. C. Tan, and A. Tay, "Online adaptive controller for simulated car racing," in *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2008, June 1-6, 2008, Hong Kong, China*, 2008, pp. 2239–2245. [Online]. Available: https://doi.org/10.1109/CEC.2008.4631096

[30] H. D. Thang and J. M. Garibaldi, "A novel fuzzy inferencing methodology for simulated car racing," in *IEEE International Conference on Fuzzy Systems, Hong Kong, China, 1-6 June, 2008, Proceedings*. IEEE, 2008, pp. 1907–1914. [Online]. Available: https://doi.org/10.1109/FUZZY.2008.4630630

[31] S. G. Varun Kumar and R. Panneerselvam, "A study of crossover operators for genetic algorithms to solve VRP and its variants and new sinusoidal motion crossover operator," *International Journal of Computational Intelligence Research*, vol. 13 (7), pp. 1717–1733, 2017.

Using a Surrogate Model of Gameplay for Automated Level Design

Daniel Karavolos
Institute of Digital Games
University of Malta
Msida, Malta
daniel.karavolos@um.edu.mt

Antonios Liapis
Institute of Digital Games
University of Malta
Msida, Malta
antonios.liapis@um.edu.mt

Georgios N. Yannakakis
Institute of Digital Games
University of Malta
Msida, Malta
georgios.yannakakis@um.edu.mt

***Abstract*—This paper describes how a surrogate model of the interrelations between different types of content in the same game can be used for level generation. Specifically, the model associates level structure and game rules with gameplay outcomes in a shooter game. We use a deep learning approach to train a model on simulated playthroughs of two-player deathmatch games, in diverse levels and with different character classes per player. Findings in this paper show that the model can predict the duration and winner of the match given a top-down map of the level and the parameters of the two players' character classes. With this surrogate model in place, we investigate which level structures would result in a balanced match of short, medium or long duration for a given set of character classes. Using evolutionary computation, we are able to discover levels which improve the balance between different classes. This opens up potential applications for a designer tool which can adapt a human authored map to fit the designer's desired gameplay outcomes, taking account of the game's rules.**

***Index Terms*—deep learning, surrogate model, artificial evolution, procedural content generation, computational creativity**

I. Introduction

Despite a long history of procedural content generation (PCG) in the game industry, there is still a gap between the algorithms developed in the academic community and those applied in the commercial sector. In an effort to address one of the major concerns of the industry, there is an increasing research interest in giving designers more control over the generated content. There are several ways to ensure that a designer can influence the creative process, e.g. by creating a set of modular components that can be recombined via a generative grammar [1] or evolution [2], by placing constraints which can be solved via answer-set programming [3], or by adjusting the objectives of a search-based generator [4].

While there has also been work towards making the designer a part of the generative loop itself, e.g. via interactive evolution or through a mixed-initiative process [5], this paper will focus on tools that allow the designer to customize the generative space a priori. This can be done with visualizations that show the effect of generative parameters on the expressive range of the generator [6], allowing designers to more efficiently tune the algorithm's parameters to the intended outcomes, or by allowing designers to literally set targets (or acceptable ranges) for their desired output. Both search-based and constrained programming approaches can use such a designer-specified goal to bias their generative output.

This paper proposes a tool for designers to create or adapt levels towards specific gameplay outcomes. The tool takes advantage of a model of gameplay that can predict emergent gameplay properties from the level structure as well as the game's ruleset. This model, trained via deep learning on a large corpus of simulated plays with artificial agents, acts as a *surrogate model* [7] and bypasses the need for computationally expensive simulations. As a design tool, the quick response times afforded by this model can be used both for immediate feedback while the designer adapts the level or the game rules (both of which affect the gameplay outcomes). Moreover, as explored in this paper, the surrogate model can be used to generate new levels or variations of existing levels through a search-based process which uses the surrogate model instead of computationally expensive simulation-based evaluations.

The use case in this paper is competitive first person shooter (FPS) gameplay, which has easily quantifiable gameplay outcomes, i.e. gameplay balance (whether a player wins easily over another) and match duration (whether the match is over quickly). The game rules and level structures that are being provided as input to the surrogate model are the game's competing character classes and the top-down view of the FPS map. *Character classes* are common in shooter games: each competing player chooses a class, which may have a different survivability and speed as well as a signature weapon which fits its role (e.g. scout, sniper class). The FPS map itself has multiple floors, to allow for FPS level patterns such as sniping positions [8]. Experiments in this paper focus on the designer-guided aspect of this framework, allowing designers to specify a map they would like to improve, the target character classes that this map is intended for, and the desired game balance and duration. The system then adapts the level via recombination and mutation, to minimize the distance with the designer's desired gameplay outcomes, using the surrogate model to predict the evolving maps' gameplay properties. A thorough evaluation with a broad set of character classes and target match durations shows that the generative approach can improve the designer's map to better match the gameplay outcomes and better take advantage of specific classes' strengths and weaknesses.

978-1-5386-4359-4/18/$31.00 ©2018 IEEE

II. Related Work

Evaluation of game content has been a topic of broad academic inquiry, e.g. via heuristics, constraints, or a combination of the two. Game levels have often been parsed in terms of their geometric or path properties to estimate a modicum of balance between two or more competing players [9]–[11]. These heuristics observe the level structure alone, and are expected to be accurate only if the two players have the same in-game characteristics (e.g. available units, tech tree, or weapon power) and the same playstyle or skill. A more objective way of assessing game balance is through agent-based simulations or actual player traces. For the latter, interactive evolution can be used indirectly to adapt a level based e.g. on the combat time of a player [12]. For the former, artificial agents have often been used in simulations to assess each and every individual in the evolving population; this approach is computationally expensive and forms the bottleneck in simulation-based evaluations for search-based PCG [4]. Such agents are often simplistic (including those used in this paper), but agents may use more general gameplaying algorithms such as Monte-Carlo Tree Search [13] or may have different goals depending on the play persona they are trying to emulate [14]. Our proposed framework attempts to alleviate the computational cost of extensive simulations by using a model trained on a broad variety of level structures and diverging in-game characteristics.

This paper explores how machine learning can be used for procedural content generation as a surrogate model, indirectly influencing the fitness function of a search-based PCG algorithm [4]. So far, machine learned models are primarily used to directly manipulate game content [15]. For instance, neural networks have mostly been used to learn level patterns which are then applied directly to the level. For example, a recurrent neural network that predicts sequences of tiles is used to create levels for *Super Mario Bros.* (Nintendo 1985) in [16]; a convolutional neural network (CNN) is used to place resources on a pre-made *Starcraft II* (Blizzard 2010) map [17]. Other work has used autoencoders to learn patterns in *Super Mario Bros.* levels and use the encoding-decoding sequence to repair broken segments [18]. Finally, CNNs have been used to predict various characteristics of *Super Mario Bros.* levels based on player annotations [19], but these networks were not used for content generation. While machine learning has a long history in procedural generation [15], the proposed framework uses its learned model indirectly (i.e. to guide evolution) rather than directly. More importantly, it is the first attempt at using a model that has learned to combine both game rules (in the form of class parameters) and level properties for actual generation. The model combines three different facets of games, i.e. game design, level design and gameplay as discussed in [20]. The framework is thus the first step towards game facet orchestration where all facets of games are considered as a whole rather than e.g. considering only the structural parts of the level [10] or the properties of weapons [21] in a vacuum.

III. Game Framework

This paper uses first-person shooter (FPS) games as a use case for mapping level structure and game parameters with gameplay outcomes; it uses that mapping to generate levels appropriate for a one versus one deathmatch game between two players. The players are assumed to be on an equal skill level, and each of them controls one avatar that belongs to a specific *character class*. Character classes are common in shooter games such as *Team Fortress 2* (Valve 2007) and have different gameplay styles and strategies, as well as a different signature weapon. This paper uses the same character class names and attributes from *Team Fortress 2* (TF2) as a benchmark for the level generator capabilities.

Deathmatch games are competitive: the player who score more kills on their opponent(s) is the winner. A session in a deathmatch game finishes usually after a specific time has elapsed or a specific number of kills by one or both players. The framework in this paper considers matches to be complete when a total of 20 kills is scored; a time limit of 600 seconds is also in place, but results from matches that timed out are ignored. The game used for these experiments is implemented in *Unity 2017*, a commercial game engine, and is based on an existing toolkit[1]. The two competing players start the game in opposite corners of a game level (described below).

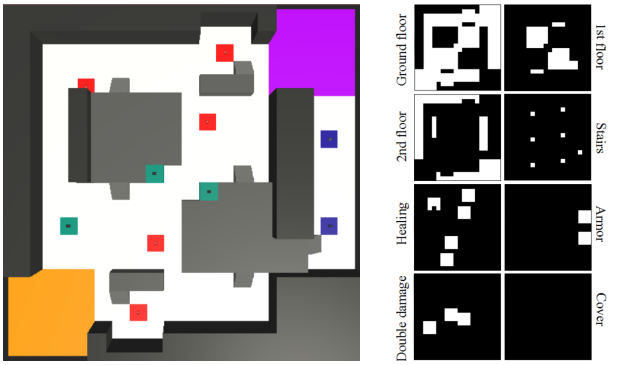
A. Game Parameters

The character class of each avatar is represented by eight parameters. Two of these parameters are specific to the character, i.e. hit points and movement speed, while the other six are characteristics of their weapon: damage (per shot), accuracy (i.e. the size of the cone in which bullets are fired), rate of fire, clip size, the number of bullets per shot and weapon range. As noted earlier, the inspiration for the class parameters is from the TF2 game; experiments in this paper use parameter values from the game itself. The one addition to TF2 parameters was that of range, to discern when AI agents should shoot.

B. Map properties

The maps in the game consist of a grid of 20×20 tiles, which can have three degrees of elevation. The ground floor and first floor are both traversable, while the second floor is inaccessible and acts as a barrier. Each tile only has one elevation (there are no tunnels or bridges). Players travel from the ground floor to the first floor via stairs; they can go down to the ground floor via stairs or drop off a ledge. An example map is shown in Figure 1a, where the floors are indicated in white, dark gray and black, and stairs between the ground and the first floor in light gray. The spawn point of the first player (P1) is always in the bottom left corner, while the second player (P2) always spawns in the top right corner. Furthermore, the maps can contain three types of pickups that are common in shooter games: a healthpack (increases health up to a maximum), armor (offers additional health which is depleted first) and a damage boost (player's bullets temporarily deal double damage).

[1] http://opsive.com/assets/DeathmatchAIKit/



(a) 3D shooter level (b) CNN inputs

Fig. 1: A view of the in-game 3D level and its transformation into CNN inputs. Orange and purple areas are the bases of player 1 and 2 respectively. Red tiles are healing locations, blue and turquoise tiles are armor and double damage powerups respectively.

IV. A Surrogate Model of Gameplay

In order to learn a mapping between levels, game rules, and the gameplay outcomes, a deep learning approach was used. This Section describes the corpus used to learn patterns from, the machine learning approaches and their performance.

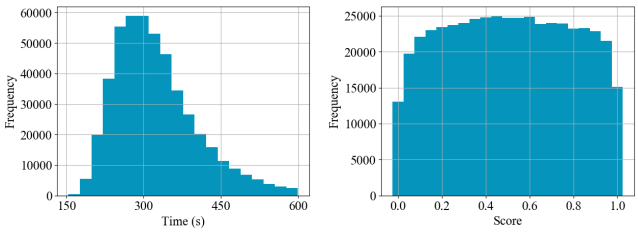
A. Creating a Game Level Corpus

In order to provide a rich and expressive dataset to learn patterns from, gameplay outcomes were collected from simulations between artificial agents playing through a broad set of levels and character classes. Agents' behavior is controlled by behavior trees that were adapted from the original toolkit.

Agents played two matches in a unique combination of class pairs (one character class per player) and levels: each agent used the same character class twice, one starting from the bottom left corner of the level and the other starting from the top right corner of the level (orange and purple in Fig. 1). In total, 10^5 levels and class pairs were generated, resulting in $2 \cdot 10^5$ data points regarding gameplay outcomes.

To generate a pair of character classes, 16 game parameters (8 per player) were normalized to a predetermined value range and a floating point value for each was randomly assigned; range was chosen randomly between short, medium and long.

To generate the game levels, a constructive map generator was implemented, using digging agents and generative grammars [22]. The overall layout of the map is created in a lower resolution representation (4×4 grid of cells), crafting two paths between player bases: one on the upper side and one on the lower side of the diagonal. Agents then operate on each cell (which is 5×5 tiles in the final level) to connect the cell to its adjacent ones. The "walls" that remain from this process are transformed into first or second floor tiles randomly; cellular automata then add more first floor tiles. The algorithm places stairs on eligible tiles with a 20% probability. Each unreachable first floor tile is transformed into a second floor, in order to guarantee traversability of first floor tiles.



(a) Game Duration (b) Kill Ratio

Fig. 2: Distribution of gameplay outcomes in the corpus.

After the level architecture is created, each cell has a 33% chance of having a pickup placed on a random tile within it.

Through the process described above, a dataset of $2 \cdot 10^5$ gameplay outcomes is created, out of which 6% are omitted as the matches are not completed within the time limit. For the remaining matches, the core gameplay outcomes that can be learned is the kill ratio (KR) of one player (P1) to the total kills, and the duration of the match in seconds. The distribution of these two gameplay outcomes is shown in Fig. 2. There is an almost uniform distribution between balanced matches (KR of 0.5) and matches where P1 had a clear advantage (KR near 1) or disadvantage (KR near 0). On the other hand, duration is skewed towards values around 300 seconds, with very few matches lasting below 200 seconds (2%) or over 500 seconds (4%). This may affect the accuracy of the deep learning approach, as will be discussed in Section IV-D.

B. Data Input

Several steps are taken to process the data collected from simulated matches in order for the machine learning approach to read it. For gameplay outcomes (the intended outputs) the kill ratio and duration are normalized to the $[0, 1]$ range: kill ratio is already normalized while duration is min-max normalized. For character class parameters, all parameters are min-max normalized in order to generate character classes for the training set. For the game level, the grid of 20×20 tiles is transformed into multiple channels using a variant of one-hot encoding (see Fig. 1). In the channels that encode pickups, a tile with a pickup and all its adjacent tiles are 1, except second floor tiles. Pickups were given more prominence in their respective channel due to their scarcity in the level and their importance in gameplay.

C. Convolutional Neural Network Architecture

Following a broad set of preliminary experiments with network architectures, activation functions and optimization strategies, the network chosen for this task is similar to [23]. This CNN has two separate information streams, one for the map and one for the pair of character classes. The level input is passed through two blocks of convolution and max-pooling, with 8 and 16 filters respectively. The convolutions are of size 3×3 (with zero-padding), and the end-result is a flat vector of 400 features for the level. The 16 parameters of the character classes are passed to a single fully-connected layer of 8 nodes,

the output of which is concatenated to the flat feature vector of the map. Finally, this combined feature vector is connected to a single fully connected layer of 32 nodes which connects to two outputs that predict the two gameplay outcomes (kill ratio and duration). All nodes use a ReLU activation function.

D. Training Results

In order to validate the performance of our CNN architecture, baselines with several multi-layer perceptron models (MLP) were tested, as well as a perceptron and linear regression (LR). For the sake of brevity, the performance of the best MLP (with 128 neurons) is reported. Additionally, to assess the importance of each input modality, the same models are trained using only the character class parameters or only the map as input by leaving the remaining inputs at a value of 0. This paper focuses on a supervised regression task; the two accepted performance criteria for such tasks are (a) the model's prediction error and (b) how much of the variance in the data is explained by the model. The former is computed by the mean absolute error, MAE_t and MAE_{KR} for duration and score respectively. The latter is computed by the R^2 metric (with typical ranges of $[0, 1]$) for these dimensions (R^2_t and R^2_{KR}). All models in this section were trained for 30 epochs, while early stopping was used to prevent over-fitting. The results reported are based on 10-fold cross-validation.

In general all baseline models perform very similarly, with a maximum difference in error of 0.01 for both dimensions. All baseline models can fairly accurately predict the kill ratio of P1 ($MAE_{KR} = 0.09$) and, surprisingly, the CNN is only slightly better ($MAE_{KR} = 0.07$). Similarly, the explained variance is fairly high with R^2_{KR} values ranging from a minimum of 0.83 for LR to 0.91 for the CNN. This high accuracy even with simple models can be explained by the significant Pearsson correlations between score and 6 class parameters (out of 16).

In contrast, all models seem to struggle to predict game duration. Although the error is quite similar to score prediction ($0.09 \leq MAE_t \leq 0.10$ for all models), the explained variance is much lower. The perceptron and LR perform the worst predictions ($R^2_t = 0.48$), while both the MLP ($R^2_t = 0.55$) and CNN ($R^2_t = 0.57$) are somewhat better. The low mean squared error but low R^2_t values is likely due to the skewed distribution of duration in the training set (see Fig. 2b).

All in all, the CNN model can predict both kill ratio and duration more accurately than baselines, although even simple models such as the perceptron perform surprisingly well.

V. Generating Levels for Specific Game Outcomes

Our main goal is to explore how the trained surrogate model of Section IV can be used to generate a map with a specific game duration and kill distribution for a matchup between two character classes. To achieve this, a genetic algorithm is used to generate levels targeting a fitness function based on the outcomes of the surrogate model. The surrogate model is exploited by using the evolving map (transformed as per Section IV-B into readable input) and the character classes (which are specified a priori and do not change), and evaluating whether the gameplay output provided by the CNN matches some designer-specified goals. The fitness for the level is calculated based on the Euclidean distance between the vector of kill ratio and duration provided by the CNN and the designer-specified vector of intended kill ratio and duration. Evolution attempts to minimize this fitness score.

Regarding the specifics of the genetic algorithm itself, the level representation is based on tiles which contain all relevant information (rather than individual channels used by the CNN). Each tile is represented as a tuple of integers describing the elevation (0 for ground floor, 1 and 2 for first and second floor) and contents (e.g. stairs, healthpack, etc.). The players' bases are identical to the original training set (P1 has 5×5 tiles at the bottom left corner, similarly for P2 at the top right corner) and evolution can not change those areas.

A new population is created by first selecting individuals to reproduce. The fittest 10% of the population is copied to the new population unchanged (elitism). The remaining 90% is chosen via tournament selection of size 5, and then recombined and mutated. For the purposes of mutation and recombination, the level (of 20×20 tiles) is divided into a 4×4 grid of cells. Each pair of individuals has a 20% chance of producing offspring via recombination: recombination is implemented by randomly picking a cell at that position from either parent. Offspring and any unselected parents are then susceptible to mutation. Every cell of every individual has a 10% chance of being mutated by one of the following variants:

- *Move Pickup:* If a cell contains one or more pickups, one of these pickups is moved to a neighboring cell.
- *Grow Cell:* One of the following operators is chosen at random: either all ground tiles that are adjacent to a 1st floor tile transform into 1st floor tiles, or all 1st floor tiles adjacent to a 2nd floor tile transform into 2nd floor tiles.
- *Erode Cell:* Opposite of *Grow Cell*, either 1st floor tiles adjacent to ground tiles are transformed into ground tiles or 2nd floor tiles adjacent to 1st floor tiles are transformed into 1st floor tiles.
- *Place Stairs:* Adds a stair to a random ground floor tile which is adjacent to only one first floor tile (remaining adjacent tiles must be ground floor tiles).
- *Place Block:* A 3×3 block of first floor tiles and a stair is created if there is enough space on the ground floor. Any pickups in this area are moved to the first floor.
- *Dig Hole:* Within a 5×5 block of first floor tiles, the central 3×3 first floor tiles are transformed into ground tiles with a stair directed inwards. Any pickups in this area are moved to the ground floor.

If a mutation is not applicable, (e.g. if a cell does not contain pickups for the first variant), another variant is attempted until either a mutation is applied or all mutation variants are tested.

After mutation and recombination, each map is analyzed in terms of traversability in order to prevent infeasible maps. The following constraints are enforced: (a) bases must always be reachable via ground floor tiles, (b) each pickup must be reachable, (c) each first floor tile must be connected to at least one stair, (d) there must be no holes in an area

TABLE I: Trade-offs between character classes in this paper.

Class	High	Low
Heavy	health, rate of fire	speed, accuracy
Pyro	damage	accuracy
Scout	speed	health
Spy	accuracy	clip size
Sniper	damage, accuracy	rate of fire, clip size

of first floor tiles without a stair to climb out of the hole and (e) a stair must always lead to a first floor. A naive constructive algorithm repairs unreachable areas and stair placements (without changing players' bases).

VI. Experiment

Experiments in this paper target a set of gameplay outcomes for specific matchups between two character classes. Moreover, to show how the algorithm can be used as a design tool for human-authored designs, all evolutionary runs start from a well-formed map and attempt to improve it. Details of this seeding process are given in Section VI-A, while other seeds are tested in Section VI-D.

In order to demonstrate the algorithm, a set of five diverse classes was chosen from *Team Fortress 2*: Heavy, Pyro, Scout, Spy, Sniper. The major trade-offs in terms of their class parameters are shown in Table I. The Heavy, Pyro and Scout all carry short ranged weapons; the Sniper carries a long ranged weapon. The cloaking and knifing abilities of the spy are ignored, treating it as a regular, medium ranged class.

Matching all character classes against each other results in 25 matchup combinations (5 between avatars of the same class). Three target match durations were selected to cover the spectrum of possible game lengths: 200 seconds (short duration), 300 seconds (medium duration) and 600 seconds (long duration). Since the two agents have an equal skill level, the target kill ratio (for P1) was set to 0.5 (i.e. balanced kills). As mentioned above, the fitness function for evolution is the Euclidean distance from both target gameplay outcomes. Significance tests reported in the paper use $\alpha = 5\%$.

A. Starting Map

The initial population is seeded with the map shown in Fig.1a: its central area is symmetrical as opposed to the edges. The map has four healthpacks on a diagonal between the two bases and two damage boosts on the first floor at the center of the map. P1 (orange) spawns near a damage boost and P2 (purple) spawns near armor.

Based on simulations on the initial map (as part of the ground truth evaluations discussed in section VI-C), the matches on this map on average last for 266 seconds, but can be as short as 203 seconds (Heavy versus Sniper) and as long as 484 seconds (Heavy versus Heavy). The map is most suitable for Snipers and least suitable for Heavies, judging by their average kill ratio against other classes. This explains the short duration of the Sniper versus Heavy matchup, which ends in a hands-down defeat for the Heavy which manages a kill ratio of 0.2 (the worst in all matchups). All matches seem to give a slight advantage to P1 regardless of their class.

B. General Performance

Each class matchup (out of 25) and intended duration (out of 3) is provided as input and intended output respectively and the genetic algorithm attempts to improve the human-authored level in 20 independent runs. Results in this section examine the fittest evolved levels at the end of each evolutionary run, after a maximum of 100 generations (although early stopping is possible). In all cases, the initial population consists of 20 copies of the human authored map of Fig. 1. Unless explicitly noted, all metrics are calculated from the average of these 20 independent runs per matchup and intended duration.

Based on the surrogate model's predictions, the Euclidean distance from the target balance and playtime decrease over the course of evolution, dropping to an average of 0.19 from 0.42 in the initial map. As expected, evolution is more challenging for the longer durations which are rarely seen in the training set: there, the average distance using predicted values is 0.35. The easiest duration to predict is the medium duration, which is very common among matches in the training set and the average distance using predicted values is 0.09. However, these distances could be low because the model has been overfitting to a skewed training set towards 300 seconds. To verify whether the predictions are reflected in actual gameplay traces, the ground truth in terms of gameplay outcomes is calculated via simulations as described in Section VI-C.

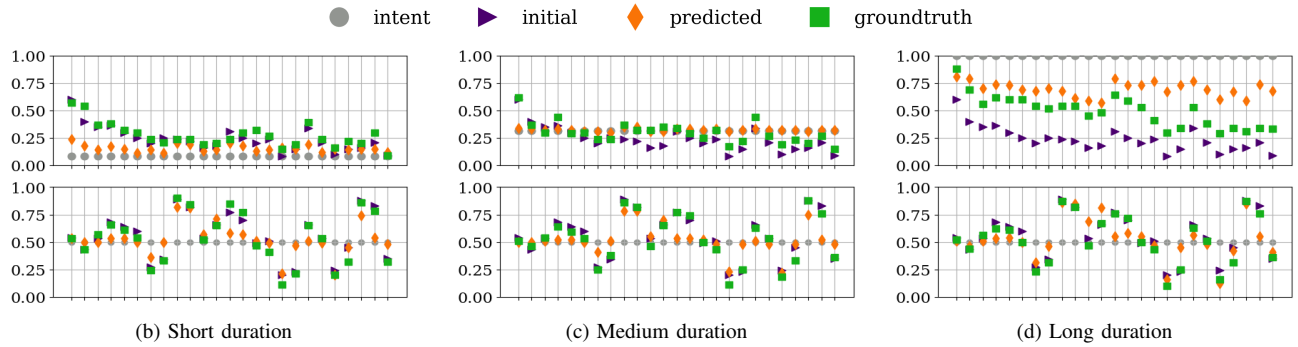
C. Improvements over the initial map

For the purposes of grounding, the best evolved maps for each run (i.e. 20 maps per class pairing and intended duration) were simulated using the agents on which the model was trained. Each map was simulated 10 times to account for the stochasticity of the AI, and gameplay outcomes (kill ratio, duration) are averaged from those 10 simulations (*ground truth* or GT). The different ground truth, predicted, and initial values per matchup are shown in Fig. 3. Two key performance metrics will be used to compare the evolved maps to the initial map, and to assess the prediction accuracy of the model. We measure *prediction discrepancy* based on Eq. (1), i.e. the difference in the Euclidean distance between intended and predicted values and Euclidean distance between intended and GT values. We measure *improvements from the initial map* based on the difference in the Euclidean distance between the initial map's GT values and intended ones and the Euclidean distance between final map's GT values and intended ones, normalized to the former to give a ratio. This performance metric, formulated in Eq. (2), is positive if the evolved map actually approaches the intended gameplay properties in one or both dimensions compared to the initial map and negative if it actually is more distant than what the designer intended.

$$P(m) = |(dist(m_{pred}, m_i) - dist(m_{gt}, m_i)| \quad (1)$$

$$O(m) = \frac{dist(m_0, m_i) - dist(m_{gt}, m_i)}{dist(m_0, m_i)} \quad (2)$$

where m is the generated map being assessed; $dist(x, y) = \sqrt{(KR(x) - KR(y))^2 + (d(x) - d(y))^2}$ is the Euclidean distance between parameter vectors x and y; $KR(x)$ and $d(x)$ is



(b) Short duration (c) Medium duration (d) Long duration

Fig. 3: The feature values of the designer's intent, the initial map, the model's predictions, and the ground truth of the evolved maps. Top row: game duration, bottom row: kill ratio. The data is split along the 25 matchups, ordered alphabetically.

Duration	O	P
Short	-10%±18%	0.14±0.03
Medium	7%±7%	0.13±0.03
Long	28%±4%	0.23±0.05
Average	8%±7%	0.16±0.02

TABLE II: Improvement over initial map and prediction discrepancy. Values are averaged across all 25 matchups, along with their 95% confidence interval.

the kill ratio and the duration (respectively) in the parameter vector x; m_{gt} and m_{pred} is the gameplay parameter vector based on GT and CNN prediction respectively; m_0 is the GT parameters for the initial map of Fig. 1; and m_i is the intended gameplay parameter vector specified a priori by the designer.

Observing the improvement of maps evolved for medium and short duration (see Table II), we note some mixed results. On the one hand, maps evolved for medium duration show a minor improvement over the initial map, while maps evolved for short duration often have negative improvements (i.e. moving away from the desired values). It is important to note that for short durations specifically, more matchups have negative improvements (14) than positive (10). Many negative improvements were also observed for medium duration (7), but not more than positive (14). When evolving for long duration, on the other hand, evolution was far more successful in improving the maps compared to the initial state. A likely reason for this discrepancy between short and long durations is because the initial map was mostly favoring short or medium durations than long durations; improving towards short durations was therefore more challenging for evolution.

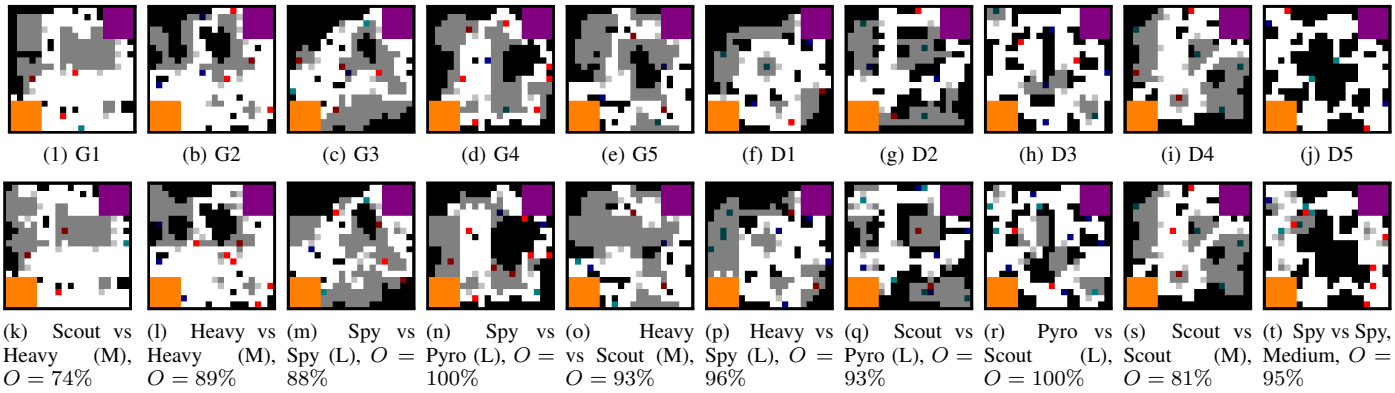
Although maps for long durations were always improved over the initial map (in all 25 matchups), the model overestimated the predicted improvements; the high P value for long durations is primarily due to the fact that most matches in truth took less time than what was predicted. Similarly, for short durations the model overestimated how short the duration would be (see Fig. 3). This is somewhat expected from a regression model (especially one with a worse R_t^2 than R_{KR}^2), but it should be noted that in the majority of matchups the ground truth durations for maps evolved for short durations were shorter than those evolved for medium (16 of 25 matchups) and maps evolved for long durations were longer than those evolved for medium (25 out of 25 matchups).

It is interesting to note that different class matchups have different performances across intended durations. The matchup with the highest improvement was Scout versus Scout for short durations (42% improvement), Sniper versus Sniper for medium durations (38% improvement), and Heavy versus Heavy for long durations (63% improvement). It is worthwhile to note that the matchups with the highest improvements were predominantly symmetrical (both players have the same class).

D. Performance with Different Initial Seeds

In order to test the generality of surrogate-based level generation, the same generation methods were applied using a broader range of levels as initial seeds. Unlike the in-depth assessment of the hand-crafted level of Fig. 1, a high-level analysis is provided using P of Eq. (1) and O of Eq. (2) as performance metrics. Ten game levels are used, each of which acts as an initial seed for 75 evolutionary runs: one run for each of 25 *Team Fortress 2* class matchups and each of three intended durations (short, medium, long). The ten levels are shown in Fig. 4: the first 5 levels are created by the same level generator used to produce the corpus, having similar but not identical patterns to levels on which the CNN model was trained on. The last 5 levels were created by a human designer, featuring a degree of symmetry and explicit level patterns such as arenas, choke points and flanking routes [8].

Results averaged across all maps of Fig. 4 for both performance metrics are shown in Table III. Results show some similar trends from Table II, and some differences. On the one hand, due to the large number of initial maps being tested, improvements were highly varied. Each initial map had a different duration and balance per class matchup and thus its improvements could be minor or negative. With generated levels as initial seeds, in particular, many matches had durations very close to the intended medium duration (causing the negative O values). It is not surprising that generated levels generally had match durations close to 300 seconds, as evidenced by the distribution in Fig. 2a. Despite



(1) G1 (b) G2 (c) G3 (d) G4 (e) G5 (f) D1 (g) D2 (h) D3 (i) D4 (j) D5

(k) Scout vs Heavy (M), $O = 74\%$ (l) Heavy vs Heavy (M), $O = 89\%$ (m) Spy vs Spy (L), $O = 88\%$ (n) Spy vs Pyro (L), $O = 100\%$ (o) Heavy vs Scout (M), $O = 93\%$ (p) Heavy vs Spy (L), $O = 96\%$ (q) Scout vs Pyro (L), $O = 93\%$ (r) Pyro vs Scout (L), $O = 100\%$ (s) Scout vs Scout (M), $O = 81\%$ (t) Spy vs Spy, Medium, $O = 95\%$

Fig. 4: Additional levels used as initial seeds for testing the generality of the method (top) and best evolved maps in terms of improvement (O), based on each initial map (bottom). Intended durations are shown as M (medium) and L (long).

Maps	Generated	Designed	All
Duration	O (improvements from the initial map)		
Short	13%±6%	24%±6%	19%±5%
Medium	-31%±47%	10%±24%	-11%±26%
Long	37%±5%	32%±6%	35%±4%
Average	6%±16%	22%±8%	14%±9%
Duration	P (prediction discrepancy)		
Short	0.13 ±0.02	0.16 ±0.02	0.15 ±0.02
Medium	0.14 ±0.02	0.15 ±0.02	0.15 ±0.02
Long	0.13 ±0.02	0.18 ±0.03	0.16 ±0.02
Average	0.13 ±0.01	0.17 ±0.01	0.15 ±0.01

TABLE III: Performance metrics averaged across 5 or 10 maps of Fig. 4 and their 95% confidence intervals.

a few extremely negative O values, in almost all cases the positive improvements were more than the negative ones per initial level and duration individually, and on average[2]. The only exception is for the 25 levels evolved for short duration based on G2 (9 positive versus 16 negative improvements). In terms of predictions, it is not surprising that using a designed level as an initial seed is more challenging than using a generated one (with slightly higher P values across durations) since generated levels have patterns closer to those learned by the CNN model. However, when averaging P values across the many evolutionary runs the differences between intended durations are less pronounced.

Fig. 4 also shows a sample of the evolved levels for a specific class pair and match duration. The evolved map with the best overall improvement (O) is shown, but it is interesting to note that not all maps are much changed from their initial states. In many cases, there are minor architectural differences while most changes are focused on the number and type of powerups (e.g. Fig. 4t and 4p). In other cases the powerups largely remain the same, but entire pathways are blocked off (Fig. 4o and 4q) or balconies or galleries introduced (Fig. 4o).

[2]There are 185 positive versus 63 negative in short matches, 173 versus 74 in medium matches, 222 versus 22 in long matches

VII. Discussion

Based on the training results of Section IV-D, it is evident that the surrogate model is able to learn several interrelations between level and class parameters. However, the biased distribution of durations in the training corpus is evidently hampering the network's ability to accurately predict values outside the medium durations around 300 seconds; this is corroborated by the lower R_t^2 value. This lack of precision in predictions affects the performance of the genetic algorithm in non-medium durations, as evidenced by higher P values in most instances of Table II and III. Results are still fairly consistent: most evolved maps have an actual shorter duration when evolving for a short duration than when evolving for a medium or long duration and vice versa. Future work should attempt to create a more fairly distributed dataset in terms of duration. Another notable improvement would be prematurely ending evolution when fitness consistently decreases compared to the initial map. In experiments of Section VI-D, the initial seed matched the intended balance and duration almost perfectly, and therefore evolution explored away from that (due to random mutations in 100 generations). Stopping evolution when maps can not be improved further would enhance results.

It is difficult to ascertain to which degree the inaccuracy of the model has led evolution away from more promising maps than the tested ones, i.e., whether it converged to a false optimum provided by the surrogate model [24]. An important strength of the surrogate model is its speed when it replaces simulation-based evaluations. Indicatively, one evolutionary run as described in Section VI lasts for 3.5 minutes on a 12-core Intel i7 processing unit; a single simulated match (optimized to run without graphics) lasts for 50 seconds on the same machine. If we use the mean gameplay metrics from 10 simulations for each class pairing to account for stochasticity (as used to calculate the ground truth in Section VI-C), one evolutionary run with 20 individuals evolving for 100 generations would last 12 days; even with a single simulation per individual one run lasts 1.2 days. This large computational overhead renders testing the surrogate model

against a pure simulation-based model unrealistic.

It should be noted that the convolutional network has been trained on synthetic playtraces, which may not always match human decision-making. Notable discrepancies when agents navigated the map of Fig. 1 was the fact that P2 rarely exploited the armor pickups of the right side of the map, which was intended as a safe hiding spot. Tactically, therefore, agents do not behave like human players do. While relying on human playthroughs for the vast data required to train deep learning models is not realistic, actual player traces from some of the promising levels can be used to fine-tune the current model.

The results of the experiments point to several possible directions for future work. First, discrepancy between prediction and ground truth values may require a more involved re-training process (with a better duration distribution and possibly human traces, as discussed above). Alternatively, we can re-introduce simulation-based evaluations when the predictive model deems that a map has sufficient quality that a more precise assessment of the gameplay outcomes (via simulations) is warranted. There are various strategies for combining simulation-based evaluations with a surrogate model, such as individual-based, generation-based or adaptive evolution control [7]. Finally, the evaluation could move away from the Euclidean distance which combines both gameplay outcomes (and treats them as equivalent) and use a multi-objective approach [25], treating distance from intended duration and distance from intended kill ratio as individual – possibly conflicting– objectives. The generator might leave this trade-off between fidelity to intended duration and fidelity to game balance up to the designer by presenting the Pareto front (and accompanying maps) to a user for manual selection.

VIII. Conclusion

This paper demonstrated how deep learning and evolutionary computation could be combined for the purposes of a generative algorithm. The neural network can process game levels as an image and game properties as a parameter vector and predict two core gameplay outcomes. These gameplay predictions can then drive the evolutionary adaptation of game levels towards specific outcomes. Focusing on the domain of shooter games, the model is able to learn the majority of patterns between levels and character classes playing in them. The interrelations between levels, class parameters and gameplay outcomes are then exploited to generate new shooter levels which target a balanced gameplay of short, medium, or long duration for a specific pairing of character classes. Future work could exploit the learned mappings among dissimilar facets to generate content of different types (e.g. character classes), generate multiple types of content simultaneously or as a real-time co-creator for a human level designer.

Acknowledgments

This research has received funding from the European Unions Horizon 2020 research and innovation programme under grant agreement No 693150.

References

[1] J. Dormans and S. C. J. Bakkes, "Generating missions and spaces for adaptable play experiences," *IEEE Transactions on CI and AI in Games*, vol. 3, no. 3, pp. 216–228, 2011.

[2] A. Liapis, "Multi-segment evolution of dungeon game levels," in *Proc. of Genetic and Evolutionary Computation Conference*, 2017.

[3] A. M. Smith and M. Mateas, "Answer set programming for procedural content generation: A design space approach," *IEEE Transactions on CI and AI in Games*, vol. 3, no. 3, pp. 187–200, 2011.

[4] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne, "Search-based procedural content generation: A taxonomy and survey," *IEEE Transactions on CI and AI in Games*, vol. 3, no. 3, 2011.

[5] A. Liapis, G. Smith, and N. Shaker, "Mixed-initiative content creation," in *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*, N. Shaker, J. Togelius, and M. J. Nelson, Eds. Springer, 2016, pp. 195–214.

[6] M. Cook, J. Gow, and S. Colton, "Danesh: Helping bridge the gap between procedural generators and their output," in *Proceedings of the IEEE Computational Intelligence and Games Conference*, 2016.

[7] Y. Jin, "A comprehensive survey of fitness approximation in evolutionary computation," *Soft computing*, vol. 9, no. 1, pp. 3–12, 2005.

[8] K. Hullet and J. Whitehead, "Design patterns in fps levels," in *Proceedings of the Foundations of Digital Games Conference*, 2010.

[9] B. Marlene, A. Aleksandr, A. Alexander, L. Christoph, N. Felix, W. Jonas, R. Marcel, R. Martin, P. Nicolas, P. Mike, and V. Vanessa, "An integrated process for game balancing," in *Proceedings of the IEEE Conference on Computational Intelligence and Games*, 2016.

[10] A. Liapis, G. N. Yannakakis, and J. Togelius, "Towards a generic method of evaluating game levels," in *Proceedings of the AAAI Artificial Intelligence for Interactive Digital Entertainment Conference*, 2013.

[11] A. Liapis, "Piecemeal evolution of a first person shooter level," in *Applications of Evolutionary Computation*. Springer, 2018.

[12] L. Cardamone, G. N. Yannakakis, J. Togelius, and P. L. Lanzi, "Evolving interesting maps for a first person shooter," in *Proceedings of the Applications of evolutionary computation*, 2011.

[13] A. Zook, B. Harrison, and M. O. Riedl, "Monte-carlo tree search for simulation-based strategy analysis," in *Proceedings of the 10th Conference on the Foundations of Digital Games*, 2015.

[14] A. Liapis, C. Holmgård, G. N. Yannakakis, and J. Togelius, "Procedural personas as critics for dungeon generation," in *Applications of Evolutionary Computation*. Springer, 2015, vol. 9028, LNCS.

[15] A. Summerville, S. Snodgrass, M. Guzdial, C. Holmgård, A. K. Hoover, A. Isaksen, A. Nealen, and J. Togelius, "Procedural content generation via machine learning (pcgml)," *arXiv preprint arXiv:1702.00539*, 2017.

[16] A. Summerville and M. Mateas, "Super mario as a string: Platformer level generation via lstms," in *Proceedings of DiGRA & FDG*, 2016.

[17] S. Lee, A. Isaksen, C. Holmgård, and J. Togelius, "Predicting resource locations in game maps using deep convolutional neural networks," in *Proceedings of the AIIDE workshop on Experimental AI in Games*, 2016.

[18] R. Jain, A. Isaksen, C. Holmgard, and J. Togelius, "Autoencoders for level generation, repair, and recognition," in *Proceedings of the ICCC Workshop on Computational Creativity and Games*, 2016.

[19] M. Guzdial, N. Sturtevant, and B. Li, "Deep static and dynamic level analysis: A study on infinite mario," in *Proceedings of the AIIDE workshop on Experimental AI in Games*, 2016.

[20] A. Liapis, G. N. Yannakakis, and J. Togelius, "Computational game creativity," in *Proceedings of the International Conference on Computational Creativity*, 2014.

[21] D. Gravina and D. Loiacono, "Procedural weapons generation for unreal tournament III," in *Proceedings of the IEEE Conference on Games, Entertainment, Media*, 2015.

[22] N. Shaker, J. Togelius, and M. J. Nelson, *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. Springer, 2016.

[23] D. Karavolos, A. Liapis, and G. N. Yannakakis, "Learning the patterns of balance in a multi-player shooter game," in *Proceedings of the FDG workshop on PCG in Games*, 2017.

[24] Y. Jin, "Surrogate-assisted evolutionary computation: Recent advances and future challenges," *Swarm and Evolutionary Computation*, vol. 1, no. 2, pp. 61–70, 2011.

[25] C. Coello Coello, G. B. Lamont, and D. A. Van Veldhuizen, *Evolutionary algorithms for solving multi-objective problems*. Springer Science & Business Media, 2007.

Learning to Play General Video-Games via an Object Embedding Network

William Woof
School of Computer Science
The University of Manchester
Manchester, United Kingdom
william.woof@manchester.ac.uk

Ke Chen
School of Computer Science
The University of Manchester
Manchester, United Kingdom
ke.chen@manchester.ac.uk

***Abstract*—Deep reinforcement learning (DRL) has proven to be an effective tool for creating general video-game AI. However most current DRL video-game agents learn end-to-end from the video-output of the game, which is superfluous for many applications and creates a number of additional problems. More importantly, directly working on pixel-based raw video data is substantially distinct from what a human player does. In this paper, we present a novel method which enables DRL agents to learn directly from object information. This is obtained via use of an object embedding network (OEN) that compresses a set of object feature vectors of different lengths into a single fixed-length unified feature vector representing the current game-state and fulfills the DRL simultaneously. We evaluate our OEN-based DRL agent by comparing to several state-of-the-art approaches on a selection of games from the GVG-AI Competition. Experimental results suggest that our object-based DRL agent yields performance comparable to that of those approaches used in our comparative study.**

***Index Terms*—artificial neural networks, deep Q-learning, reinforcement learning, computer games, general video game AI**

I. Introduction

General video-game AI (GVG-AI) is an area regarding the development of general algorithms that enable AI agents to play a wide range of different video-games with minimal tailoring to specific games. While developing techniques for General AI is a key focus of research into GVG-AI, general video-game playing agents also have a number of applications within the games industry. Asides from the obvious applications, such as a replacement to hand-coded in-game AI, GVG-AI can also either be used as a development tool or as a proxy for human play-testers. Such agents could be employed effectively in a wide array of applications from testing game balance [1] to evaluating procedurally generated content [2]. As well as applications within games and games design, GVG-AI also has wider implications for the field of AI, as techniques which work well on video-games can often also be applied to real-world problems.

One promising area in the search for general video-game players is Deep Reinforcement Learning (DRL). DRL agents have been successfully applied to a wide range of video-games ranging from 2D arcade games [3] to challenging 3D shooters [4]. These agents learn through interacting with the game autonomously, using a deep neural network to select actions based on the current state of the game. During this process the agent receives rewards (usually dictated by the in-game score) which indicate how well it is performing. By using an appropriate reinforcement learning algorithm the agent is able to modify its neural network in order to maximise this reward signal. However, such agents are far from perfect, and can sometimes be difficult to apply in practice – a problem compounded by the fact that they typically take a long time to train. A significant consideration into the design of these agents is how information from the game is presented to their neural networks (*i.e.* the representation given to the agent), as well as the design of the networks themselves.

Current state-of-the-art DRL approaches to video-games learn directly from raw video data, using deep *convolutional neural networks* (CNNs) [3]. While widely applicable, this approach is subject to limitations for certain applications. For example, many games (e.g., Starcraft) feature controllable cameras, meaning much of the game-state is obscured from the agent at any given point. Working around this, e.g., by putting the camera under the agent's control, adds additional complexity to the agent. Additionally, in many cases it may be undesirable, or even impossible to produce a video-output for the agent to consume. Rendering videos for multiple agents may be prohibitively expensive, and in some cases there may be no obvious way to produce a good visual representation for NPCs (non-player characters). More importantly, interpreting raw video data at a pixel level is substantially different from how human players appear to play, as studied in [5].

Unlike many other reinforcement learning tasks, the ground-truth information about the current state of the environment is often available in video-games, although such information needs to be organised and presented to an agent in some way. Hence, the use of this direct information about the current game-state could be alternative to working directly on raw video data. For instance, Samothrakis *et al.* [6] employ a fixed set of general features, i.e., distance to the nearest enemy, number of tokens collected and so on, to encapsulate the current game-state. While this approach often works well, those general features have to be handcrafted, which is laborious and requires human expertise. Moreover, this approach is relatively game-specific and hence generally inappropriate to GVG-AI. To overcome this limitation, various game-independent object representations, have been employed [7], [8]. In an object representation, each game-state observation is given as a list of objects, and their classes and attributes. For example, the state of one round of the

978-1-5386-4359-4/18/$31.00 ©2018 IEEE

game *Pong* might be represented by two objects of the class `bat`, with attributes of `x-coord`, `y-coord`, and `player`, and an object of the class `ball` with attributes `x-coord`, `y-coord`, `x-velocity`, and `y-velocity`. Many video-games rely on objects for their internal representation of the game-state. For instance, the popular *Unity 3D* game engine relies heavily on game objects, and even the early *Atari* console used a primitive sprite-based system. In general, the use of object representations not only leads to an effective approach to representing game-states across a wide variety of video games but also has a number of practical benefits. For example, it allows the use of different subsets of objects for different agents. Also, objects provide useful anchor points for applying various advanced reinforcement learning techniques such as hierarchical reinforcement learning [9], intrinsic motivation [10], and planning [11]. Given there are a different number of objects in different states of game, however, how to structure this information in a way that can be input into a conventional deep neural network is a key issue of using object representations with DRL. Previous solutions to this problem mimic an image representation by using an "object perception grid", where objects are overlayed onto a grid and mapped to the nearest cell determined by their `x` and `y` co-ordinates within the game. The number of objects mapped to each cell is then used as an input for a conventional neural network (either fully-connected or convolutional). Unfortunately, such a solution requires selecting an appropriate grid size manually, and entails a large input space, increasing the required neural network complexity. In general, it also still suffers from many of the same problems as encountered by using raw image representations, such as a restricted field of view.

In this paper, we present a novel approach to address the object representation issues in GVG-AI. To overcome all the aforementioned limitations, we adopt a specific type of neural network architecture, set networks [12]–[14], to develop our *object embedding network* (OEN). This network can not only take a list of object-feature vectors of arbitrary lengths as input to produce just a single, yet unified, fixed-length representation of all the objects within the current game-state, but also be trained on a given task simultaneously. Hence, our OEN-based approach provides an alternative way to apply DRL algorithms within video-games, based on object information. Our approach is generally motivated by recent advances within approaches to relational reasoning [13] and dynamics prediction [14], which suggest that working with objects, rather than raw data, can help scale up deep learning to more complicated tasks in a similar fashion to human information processing.

Our main contributions in this paper are summarised as follows:

1) We propose an OEN model, based on set networks, for learning directly from sets of object feature vectors.
2) We develop an OEN-based GVG-AI agent for playing general video games.
3) We evaluate our approach on selected games from the GVG-AI competition and demonstrate that it performs comparably to a variety of other popular approaches for representing game states.

The rest of this paper is organised as follows. Section II reviews related work. Section III presents our OEN model. Section IV describes our object-based approach to GVG-AI. Section V describes our experimental settings and reports experimental results. Finally, Section VI discusses issues and implications arising from this study.

II. Related work

A. Deep Reinforcement Learning for video-games

Reinforcement learning is a sub-field of machine learning where agents autonomously interact with an environment and seek to maximise some reward signal. *Deep reinforcement learning* (DRL) is an extension of classical 'tabular' approaches to reinforcement learning which enable these techniques to be applied to more complicated problems. By using the in-game score counter as a reward signal, DRL can be applied to develop agents for playing video-games [3], making video-games a popular test-bed for new DRL algorithms. Those algorithms work by using a deep neural network to 'score' possible actions given a particular game-state, which is then trained according to a loss function. Such a loss function for DRL may be formulated based on the agent's past experience as well as the reward obtained.

The *deep-Q network* (DQN) algorithm [3] is a pioneering work in applying DRL to general video games playing, where the DQN was trained to play a variety of games for the *Atari 2600* games console. In the original DQN algorithm, the game-state is presented to the agent as a series of four images from four consecutive frames of video output, which is interpreted by a deep *convolutional neural network* (CNN) with four input channels. The success of this DQN algorithm has led to a number of alternative DRL algorithms for video-game playing [15], [16] for the *Atari* system, and these algorithms have also been applied to a variety of other video games [4], [17]. While the DQN algorithm and its variants can be easily adapted to a variety of input types and network architectures, these mainly use the same input format; i.e. a sequence of frames from a game video stream. A notable exception is the use of object perception grids, which is described below in Section II-B.

B. Object-oriented reinforcement learning

Reinforcement learning from objects has previously been studied within object-oriented reinforcement learning [18], which allows for exploiting structural information at an object level. In object-oriented reinforcement learning, the state-space is expressed in terms of a set of objects. These objects all belong to some class from a fixed set of classes. Each object is an ordered tuple of object attributes, where the domain of these attributes is determined by the object class. For example, for a simple empty 5×5 grid-world, we might formulate the state-space with a single class, `Agent`, of attributes `x` and `y`, $\mathit{Dom}(\mathtt{x}) = \mathit{Dom}(\mathtt{y}) = \{1, 2, \ldots, 5\}$. Then the initial state s_0 would be a single object `agent` from the `Agent` class with attribute values $\mathtt{x} = \mathtt{y} = 0$.

Conventional approaches to solving these problems usually involve planning algorithms, and first order logic [19], or otherwise rely on the discrete nature of the environment, which is generally incompatible with DRL approaches. In particular, structuring this information in a way that can be used by a neural network is a challenging problem. Very recently, this

problem has been addressed via an object perception grid representation, e.g., [7], [8]. In this representation, objects are mapped onto grid squares, based on their x and y attributes. Each of these grid squares is then treated as an input neuron, which is set to 1 if an object of a given class is present at that cell, and 0 otherwise. The full representation is then given by multiple input grids, one for each possible object class. This effectively produces an image-like representation, which can be fed into a CNN. Apart from removing some of the complexity of the input, this approach suffers from most of the same problems as a visual representation. Moreover, this representation also requires the designer to select a grid size and coarsity. While many games may feature a natural choice for these, for many games it may require careful selection and additional tuning to select these parameters appropriately. Additionally this process can only be applied where objects are related by a clear 2D structure.

III. Object Embedding Network

In this section, we present an *object embedding network* (OEN) to learn a unified object representation from arbitrary sets of objects characterised by a variety of object features without being limited to 2D spatial structures. Our OEN model is based on an emerging class of deep neural-network architecture, set networks [12], which were recently developed to tackle the input data in a set form. In general, objects in a game-state naturally stand in a set form. By using the same principles behind set networks, our OEN transforms an arbitrary number of object feature vectors corresponding to a game state into a single fixed-length "*unified*" object representation. This unified representation can be used for a variety of different purposes. Our OEN can be trained to learn a unified representation and fulfil a specific learning objective simultaneously.

At game-state s_t, assume that there is a set of objects, $O^{(t)} = \{o_k^{(t)}\}_{k\in 1,\cdots,K_t}$, where each object, $o_k^{(t)}$, can be characterised by a feature vector (or a number of attributes), $\boldsymbol{x}_k^{(t)}$. Hence, the feature vectors of all K_t objects collectively form a set, $X^{(t)} = \{\boldsymbol{x}_k^{(t)}\}_{k\in 1,\cdots,K_t}$, for game-state s_t. Our problem is how to learn a fixed-length unified feature vector that retain as much representative information conveyed by K_t objects as possible for arbitrary K_t.

A common way to get representative information of a set of vectors is to compute some statistic about the set. In practice, this can be achieved using simple arithmetic pooling functions, e.g., max or sum pooling, applied element-wise, which condense an input set of vectors into a single fixed-length vector of the same dimension. However, simply applying simple pooling functions over a set of object feature vectors is likely to incur a loss of important information. For example, if the object features consist of x and y co-ordinates then taking the mean of all object feature vectors simply ends up with the average position of all objects. While this is useful information, it does not convey the important information, e.g., "is object $o_i^{(t)}$ next to object $o_j^{(t)}$?". Hereinafter, we drop out the explicit game-state index, t, to facilitate our presentation.

Motivated by set networks [12], we deal with this problem by embedding raw feature vectors of objects into a higher dimensional space, which allows for retaining non-trivial information after pooling. This can be achieved by applying a proper "embedding" function E to the feature vector of each object:

$$E(X) := \{E(\boldsymbol{x}_1), ..., E(\boldsymbol{x}_K)\}.$$

Let Π to denote a pooling function, a unified representation of the object feature set, X, is then achieved by

$$\boldsymbol{r}(X) = \Pi_{k\in 1,\cdots,K} E(\boldsymbol{x}_k).$$

However, finding a proper embedding function explicitly is extremely difficult in general. Also, the optimal choice of such an embedding function is task-dependent. Instead of using an explicit embedding function, we can employ a neural network to learn an optimal embedding function. Furthermore, for a specific task based on the unified representation, we can incorporate another neural network for fulfilling the given learning objective and learning the optimal embedding function simultaneously.

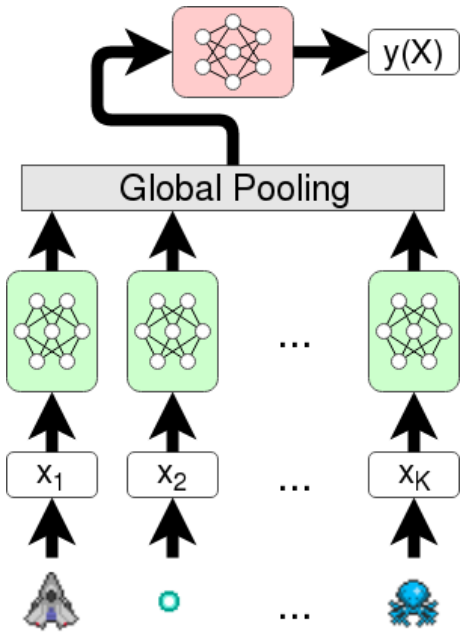


Fig. 1: Basic object embedding network architecture.

As illustrated in Fig. 1, a generic object embedding network (OEN) consists of *embedding network* (shown in green), *global pooling function* and *task network* (shown in pink). For a set of K objects in a game-state, K identical embedding networks $E(X;\theta_E)$ are employed for embedding different objects, respectively, where θ_E is a collective notation of parameters shared by all K embedding networks. The global pooling function condenses the embedding object representations produced by K embedding networks to yield a fixed-length unified object representation: $\boldsymbol{r}(X) = \Pi_{k\in 1,\cdots,K} E(\boldsymbol{x}_k, \theta_E)$. Then, the unified representation is fed to the task network denoted by $P(\boldsymbol{r}(X);\theta_P)$ where θ_P is a collective notation of parameters in the task network.

Parameter estimation in the OEN is done by optimising a loss function defined on training data, $\mathcal{D}$, given for a specific

task, $L(\mathcal{D}; \theta_E, \theta_P)$:

$$\{\theta_E^*, \theta_P^*\} = \text{argmin}_{\theta_E, \theta_P} L(\mathcal{D}; \theta_E, \theta_P).$$

For some loss function L, e.g., in a supervised learning task, a prediction-error based loss function can be used. In Section IV, we detail a loss function defined on transitions (s_t, a_t, r_t, s_{t+1}), drawn from experience-replay, for our reinforcement learning tasks.

Contextual information regarding the relationship between an object and other co-occurring ones at the same game-state can play an important role. We can exploit this by replacing our embedding function $E(\boldsymbol{x})$ with a "contextual" embedding function $E(\boldsymbol{x}, X)$ which takes into account information from the wider set when embedding each object. To this end, different techniques have been proposed in set networks, e.g., [12], [14], [20], to explore this contextual information. Motivated by the work of [12], [20], we adopt a simple global-context based method to explore the contextual information in our work. In this method, some statistic Π of feature vectors of all the objects in the set X is first estimated by $\hat{\boldsymbol{x}} = \Pi_{\boldsymbol{x} \in X} \boldsymbol{x}$. Then, the feature vector of each object, $\boldsymbol{x}$, is concatenated with this statistic vector, $\bar{\boldsymbol{x}}$, to form a "contextualised" feature vector of the object: $(\boldsymbol{x}, \hat{\boldsymbol{x}})$. Instead of the feature vector of each object, $\boldsymbol{x}$, its contextualised feature vector, $(\boldsymbol{x}, \hat{\boldsymbol{x}})$, is fed to the embedding network in the OEN. In particular we adopt the same "equivariant" transformation proposed in [12] which is given by:

$$f_{equiv}(\boldsymbol{x}, X) = \boldsymbol{x} - maxpool(X).$$

By using this method, our OEN model can be extended to explore contextual information without altering its general architecture and learning algorithms.

IV. Model Description

In this section, we present our method for establishing an OEN-based DRL agent for playing general video games. We first describe object and feature extraction required by the OEN and then propose our OEN implementation and its deep Q-learning algorithm.

A. Object and feature extraction

For our agent, the process of identification and extraction of objects is handled by the environment. That is, we assume that object extraction can be done directly via access to the ground truth of the environment. Hence, the wide-spread use of object-oriented programming languages should help with this process, as many objects are likely to be treated as such in code. Thus, the description of a game state is given in an object-oriented format; i.e., observation O from a game-state is given as a list of objects, $O = \{o_1, \cdots, o_K\}$.

Given a list of objects in this format, we still need to characterise those objects via a number of attributes to meet our requirement of our OEN-based DRL. In essence, this is a feature extraction process to obtain object feature vectors from the raw objects basted on their attributes, which leads to a pre-processing function `process_observation` required by our OEN. A natural solution to feature extraction is concatenating all real-valued attributes of an object into a single feature vector. However, this solution results in a problem; while our OEN can handle only fixed-length object feature vectors, the number of attributes used to characterise an object is not fixed and different objects could have a different number and types of attributes. Hence, the user must select a set of attributes applicable to all the objects for a fixed-length feature vector. In this manner, a set of fixed-length feature vectors, $X = \{\boldsymbol{x}_1, \cdots, \boldsymbol{x}_K\}$, can be extracted for a list of objects, O, in a game-state. This set of feature vectors are fed into our OEN, which acts as a value-network for the agent.

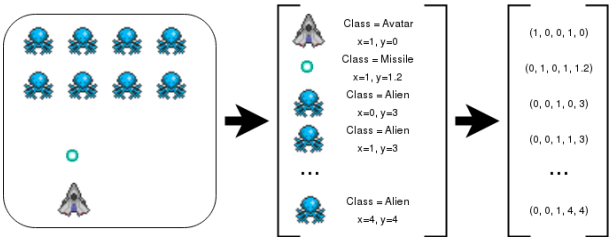


Fig. 2: Exemplar object feature extraction process.

Fig. 2 depicts an exemplar object feature extraction process. As seen in Fig. 2, a game-state is broken down into a (finite) set of objects, along with a number of attributes for each object, as chosen by the user, e.g., position, class of object, and so on. This list of objects is then converted into a list of feature vectors by mapping each object to a fixed-length vector based on its attributes. In this example, a one-hot vector of the object class concatenated with the object's co-ordinates yields a 5-dimensional object feature vector.

Algorithm 1 Q-learning algorithm for OEN

// Initialise agent
initialise empty replay memory M
initialise θ for OEN Q-function $Q(s, a; \theta)$
$\theta_{target} \leftarrow \theta$
// Initialise environment
`env.reset()`
`step` $\leftarrow 0$
while `step` $\leq$ `max_step` **do**
  `step` $\leftarrow$ `step` $+1$
  // Get action from Q-function
  $s_t \leftarrow$ `env.get_state()`
  $O_t \leftarrow$ `process_observation`(s_t)
  $a_t \leftarrow$ `epsilon_greedy`$(Q(O^t, a; \theta))$
  // Step environment and observe result
  $r_t, s_{t'} \leftarrow$ `env.apply_action`(a_t)
  $O_{t'} \leftarrow$ `process_observation`$(s_{t'})$
  $T_{t'} \leftarrow$ `env.has_ended()`
  add tuple $(O_t, a_t, r_t, O_{t'}, T_{t'})$ to M
  // Train network
  sample tuple $(O'_t, a'_t, r'_t, O'_{t'}, T'_{t'})$ from M
  $Q_{targ} \leftarrow r'_t + T'_{t'} \cdot \gamma \cdot \max_a Q(O'_{t'}, a; \theta_{target})$
  update θ via gradient descent on $(Q(O'_t, a'_t; \theta) - Q_{targ})^2$
  if `step` $\%1000 == 0$ **then**
    $\theta_{target} \leftarrow \theta$
  end if
end while

B. OEN-based Q-learning

Motivated by the deep set network of [12], we develop an object embedding network to implement our DRL agent,

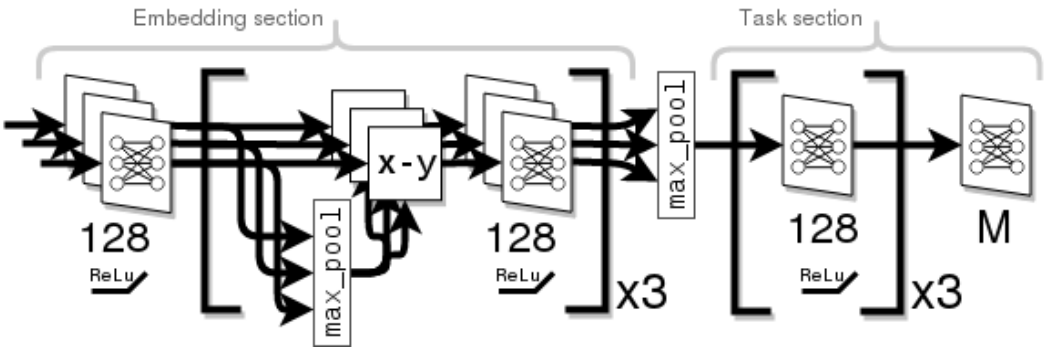


Fig. 3: The object embedding network used to implement our DRL agent.

as illustrated in Fig. 3. In this OEN, the embedding network consists of four layer of 128 ReLu units, with an "equivariant" transformation $f_{equiv}(\boldsymbol{x}, X) = \boldsymbol{x} - maxpool(X)$ between each layer. To generate a unified object representation, all embedding representations are pooled by element-wise max pooling across the whole set. The task network consists of three fully connected hidden layers of 128 ReLu units, and a final output layer of M linear units corresponding to value functions of M possible actions used in playing the given video games. It is worth mentioning that our OEN-based DRL implementation is largely identical to the DQN-based DRL [3] apart from two aspects: a) we use the OEN shown in Fig. 3, while the DQN uses a deep CNN as a learning model, and b) our OEN works on object feature vectors, while the DQN works on raw video data. Thus, the same deep Q-learning algorithm can be adapted to train our OEN-based DRL agent.

In a Q-learning based DRL algorithms, the Q-values are estimated using a value function given by a neural network $Q(s, a; \theta)$, where s is a game-state, a is a selected action and θ is a collective notation of all the parameters in this neural network. The policy for the agent is then given by selecting the action which maximises $Q(s, a; \theta)$ for the agent's current state. In the DRL learning algorithm proposed by Mnih *et al.* [3], the DQN actually outputs a value vector, $\boldsymbol{Q}(s; \theta)$, for all the actions simultaneously, where $Q(s, a_m; \theta)$ is the mth element of this output vector, reflecting the value of the mth action. Given a sequence (s_t, a_t, r_t, s_{t+1}), we can obtain an estimate for $Q(s_t, a_t)$ using the Bellman equation:

$$\hat{Q}(s_t, a_t) = r_t + \gamma \max_a Q(s_{t+1}, a).$$

This estimate is used as a target Q_{targ} for $Q(s_t, a_t)$. Thus, we define the Q-learning loss function at game-state t as follows:

$$L(\mathcal{D}_t; \theta) = (Q(s_t, a_t; \theta) - Q_{targ})^2,$$

where $\mathcal{D}_t = (s_t, a_t, r_t, s_{t+1})$ is training data retrieved from an experience-replay memory [3]. Since the new estimate Q_{targ} depends heavily on the previous values $Q(s_{t+1}, a; \theta)$, a separate network parameterised with the known θ_{target} is used to obtain Q_{targ} estimates. θ_{targ} is then updated once the new parameters θ in the OEN are achieved during the learning. To optimise the loss function for the Q-learning, we employ the Adam optimizer [21], a gradient-based optimiser. Network parameter update is also done in mini-batches of multiple state-actions and targets simultaneously. Since multiple sequences of different lengths are not readily expressible as fixed-size tensors (which is required by most deep-learning libraries), for each batch we pad each sequence with zero vectors until they have the equal length. A mask of these zero elements is produced and used to nullify their contribution to the output of the OEN. For clarity, we describe the detailed Q-learning algorithm used for training our OEN in Algorithm 1.

V. Experiment

In this section, we evaluate the performance of our OEN-based DRL agent on five selected games used in the GVG-AI competition [22] by comparing with two baseline agents that use different representations of the game-state, and measure average performance during training on a variety of games. To ensure a fair comparison between the different forms of representations, for each agent we change only the agent's neural network and the format of the observation presented to the agent, keeping the rest of the agent design the same.

A. Test environment

In order to adequately test our approach, we require a corpus of distinct video-games, preferably unified under a single framework. A common choice for this is the *arcade learning environment* (ALE) [23], however, the ALE does not provide access to ground truth information about object attributes, hence we instead look to the GVG-AI Competition.

The GVG-AI competition [22] is a regular competition challenging researchers to build AI agents capable of playing a variety of different video-games. These games are specified via *video-game description language* (VGDL) [24] where games are defined by a block-based sprite system, and are often based on well-known titles. Importantly, games from previous rounds of the competition are released to the public, proving a large collection of games, all running within the same framework. See Figure 4 for some examples of games from the GVG-AI Competition.

Importantly, information about the current game-state is presented to the agent in the form of a state-observation which includes a list of information about the various sprites (i.e., game objects) within the game. Additionally, while not explicitly provided to the agent, a video output is also produced for human consumption.

As well as giving direct access to in-game objects, VGDL provides a number of other benefits as a reinforcement learning test-bed, including:

(a) (b) (c) (d) (e)

Fig. 4: Five games from Test Set 1 of the GVG-AI Competition used in our experiments: (a) Aliens. (b) Boulderdash. (c) Missile Command. (d) Survive Zombies. (e) Zelda. Missile Command, Boulderdash and Zelda are based on classic arcade games of the same names, while Aliens is loosely based on the game *Space Invaders*.

- Existing tasks can easily be modified to test how this affects the agent.
- New games/tasks can be quickly synthesised for particular purposes.
- A large available pool of pre-existing tasks in the form of GVG-AI competition games.
- Direct access to the underlying mechanics and ontology, which may be useful as a ground-truth for investigating things such as model-based agents and transfer learning.

For our environment we adapted the original `py-vgdl` code [1], adding in some extra functionality from the competition version such as support for sprite images. Another option would have been to use the code provided for the GVG-AI Competition, although when we started this work there was no python client available.

B. Experimental Settings

To evaluate the performance of each agent we train the agent for 2,000,000 steps, testing the agent for 50 episodes (without training) every 50,000 steps, and record the reward obtained over each of these episodes. While there are a number of different indicators of agent performance, e.g., percentage of games won, we select average episode rewards as this most closely reflects the reinforcement learning objective of the agent, hence is less sensitive to factors such as a mis-specified reward. Additionally, different users may have different criteria for how they want agents to perform during training, i.e., some may be interested in short-term performance after a certain number of training steps, while others may be interested only in the 'asymptotic' final performance, hence we record agent performance throughout training.

We modified our environment code to included support for three different forms of observation types:

1) Image representation: A sequence of raw pixel images of the game screen, appropriately sized to be close to the 84×84 post-processed resolution of the original DQN algorithm.
2) Object representation: A list of game objects, each given in the form of vector with: a one-hot vector of object class, object co-ordinates, object orientation, and values of given object resources.
3) Feature representation: A list of the shortest distances from each object class to the player avatar, plus a list of any additional avatar resources. This is the same as the features described in [6].

In order to simplify our input space, and since certain objects in the game are irrelevant (or invisible) to the agent, for each game we defined a list of the important object classes, and ignore any objects from classes not on that list for both the objects and features representation. We also remove the frame-skip functionality from our agent as GVG-AI Competition games have a slower update rate, so being able to select actions at each step is important.

For each of these three observation types, we modify our baseline agent as follows, giving us three different agents[2]:

1) For the image representation we use the same CNN as used in [16] with a final linear layer of M outputs. Similar to the original DQN algorithm we also compile states from two consecutive frames (we do not use four frames as sprites are visible every frame in VGDL which is not the case for certain ALE games[3], and this reduces computational burden).
2) For the object representation we use the OEN described in Figure 3.
3) For the feature representation, we use a simple fully-connected neural network with layers of 64, 64, 128, and M ReLu units, respectively.

Where M is the number of possible actions in the given environment. All the agent hyper-parameters use in our experiments are as follows: γ=0.99, ϵ-start=0.5, ϵ-final=0.1, ϵ-anneal-step=500,000, replay-memory-size=50,000 learning-rate=0.00025, mini-batch-size=32, the agent is trained every four steps, and the parameters are initialised randomly with a Gaussian distribution: $\mathcal{N}(0, 0.1)$.

We select five games from Test Set 1[4] of the GVG-AI competition as our test set: Aliens, Boulderdash, Missile Command, Survive Zombies, and Zelda. For each of these we use the first level (i.e., `level_0`) as our game environment. We train our agents on each game four times, using a different seed for agent initialisation each time. Initial environment seeds are reset to the same value for each agent, ensuring that there are no differences in agent performance due to different environment initialisations.

C. Results

Full results of our experiments are shown in Figure 5. We also report the best mean test score on each game for each agent in Table I, as these give an idea of the theoretical max

[1] Available here: https://github.com/EndingCredits/gym_vgdl

[2] Our agent & experimental code is available at: https://github.com/EndingCredits/Object-Based-RL

[3] Due to the limited sprite buffers of the *Atari* console, a common optimisation is to draw certain sprites only every other frame.

[4] Found here: http://www.gvgai.net/training_set.php?rg=1

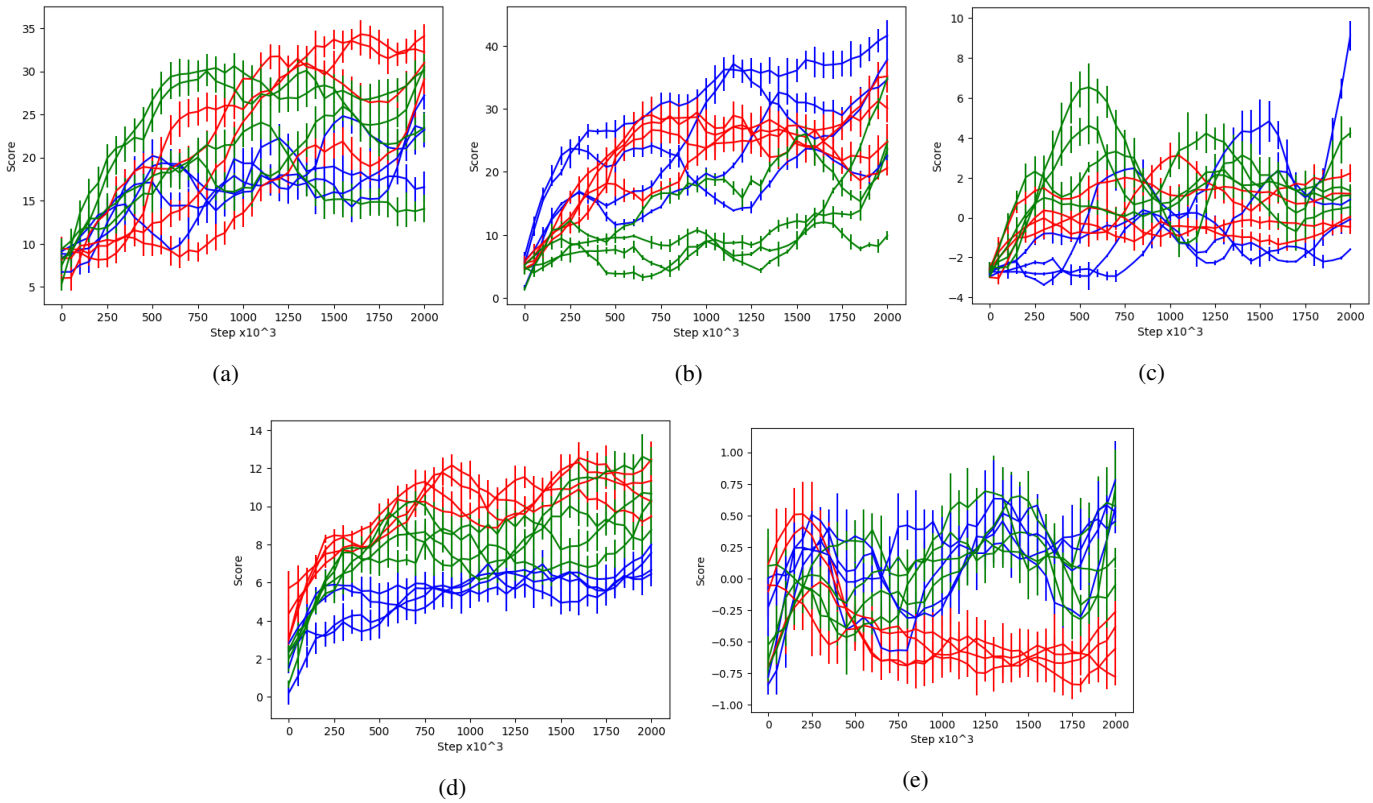


Fig. 5: Average episode rewards over 50 episodes at various points during training for five games: (a) Aliens. (b) Boulderdash. (c) Missile Command. (d) Survive Zombies. (e) Zelda. Results are are smoothed using a forth-order Savitzky-Golay filter with a window size of 21 to improve readability. Lines in Blue are agents with feature representation & fully connected network, Red are image representation & CNN, and Green are object representation & OEN (ours). Best viewed in colour.

TABLE I: Best mean score for each agent over 50 episodes.

	Aliens	Boulderdash	Missile Command	Survive Zombies	Zelda
Image	35.98	41.90	5.72	**16.40**	1.16
Features	29.82	**43.34**	9.12	10.42	1.58
Objects (ours)	**37.30**	35.90	**10.44**	14.88	**1.68**

performance of each agent type accounting for variability in agent parameters (although clearly these results are subject to sample bias, and are likely to be overestimates).

Due to the unpredictable nature of deep reinforcement learning we observed a large variance in agent performance between episodes but also between the average of different tests, making it difficult to compare individual agent results. Additionally, due to time and computational constraints we were only able to train for two million steps (comparable to eight million frames with frame-skip). This is significantly fewer than the tens and hundreds of millions of frames which many agents from the literature are trained for, meaning the results reported here may only be representative of the early stages of training. Nevertheless, we do observe certain patterns. In particular, there are notable difference in performance between representations. This is not surprising, as it is well known that choice of representation and network architecture has a big impact on performance across other areas of deep learning. However, this difference in performance is not consistent across all games; different games seem to favour different representations. Surprisingly, the "features" baseline outperforms each other agents on certain games, despite often obscuring information about the game state (for example, the agent is given distances to certain objects, but not the direction to them, or the number of them).

It is observed from all the experimental results reported above that our object-based agent is capable of learning in all five games we tested it on. Additionally, across all the games, our agent performs comparably with the other two approaches. To this end, our experiments demonstrate that our OEN-based DRL agent can be an effective alternative to the existing agents for playing general video games.

VI. Discussion

While we believe out method is generally widely applicable, a fundamental assumption made for our approach is that the game-state can be expressed in terms of objects. In some games object information may be unavailable, thus our approach cannot be applied in those games. Nevertheless, our object-based approach could work on those games, e.g., Starcraft, where image-based approaches, e.g, DQN [3], are difficult to apply due to unavailability of a complete visual-output snapshot of the full game-state (e.g. due to presence of a controllable camera, or mouse-over options). In general, object extraction from a game-state can be done either directly via access to the ground-truth of the environment or from video or some other sources. When the ground-truth of the environment is available, object extraction is usually straightforward by utilising the object-oriented nature of most games (although this may also rely on certain domain knowledge to identify

which objects are relevant to the agent). Otherwise, objects could be extracted directly from video data based on the state-of-the-art semantic image segmentation techniques. As a game environment is usually much simpler than natural images, existing semantic image segmentation techniques should be sufficient for this task. Another limitation of our approach is that it requires the user to find a fixed number of attributes to form feature vectors for all objects applicable to the OEN. In future this requirement for a fixed number of attributes across all object classes could be removed by pre-embedding all objects into a fixed-size space, or using class-specific embedding functions.

The use of relational information between objects is important when expressing the game-state. Indeed, Liang *et al.* [25] showed that simple relationships between objects form a good feature set for reinforcement learning in video-games. In our work, we use only a simple method to exploit contextual information which has limited ability to capture these relations. In set networks, there are more sophisticated methods to exploit the contextual information, e.g., those used in [13], [14]. To achieve more effective unified object representations, our OEN model described in Section IV-B can be improved by adopting those techniques developed for set networks.

To conclude, we have presented a novel approach to learning directly from semi-structured object information via an OEN for playing general video games. A comparative study based on five GVG-AI competition games suggests that our approach yields performance comparable to two state-of-the-art approaches in general. In our ongoing research, we aim to address the issues and the limitations discussed above for improvement.

REFERENCES

[1] A. Champandard, "Making designers obsolete? evolution in game design," http://aigamedev.com/open/interview/evolution-in-cityconquest/, 2012, [Online; accessed 2016-02-18].

[2] T. S. Nielsen, G. A. Barros, J. Togelius, and M. J. Nelson, "General video game evaluation using relative algorithm performance profiles," in *Applications of Evolutionary Computation*. Springer, 2015, pp. 369–380.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[4] G. Lample and D. S. Chaplot, "Playing fps games with deep reinforcement learning." 2017.

[5] R. Dubey, P. Agrawal, D. Pathak, T. L. Griffiths, and A. A. Efros, "Investigating human priors for playing video games," *arXiv preprint arXiv:1802.10217*, 2018.

[6] S. Samothrakis, D. Perez-Liebana, S. M. Lucas, and M. Fasli, "Neuroevolution for general video game playing," in *Computational Intelligence and Games (CIG), 2015 IEEE Conference on*. IEEE, 2015, pp. 200–207.

[7] K. Kunanusont, S. M. Lucas, and D. Pérez-Liébana, "General video game ai: Learning from screen capture," in *Evolutionary Computation (CEC), 2017 IEEE Congress on*. IEEE, 2017, pp. 2078–2085.

[8] K. Narasimhan, R. Barzilay, and T. Jaakkola, "Deep transfer in reinforcement learning by language grounding," *arXiv preprint arXiv:1708.00133*, 2017.

[9] N. Topin, N. Haltmeyer, S. Squire, J. Winder, M. desJardins, and J. MacGlashan, "Portable option discovery for automated learning transfer in object-oriented markov decision processes." in *IJCAI*, 2015, pp. 3856–3864.

[10] T. D. Kulkarni, K. Narasimhan, A. Saeedi, and J. Tenenbaum, "Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation," in *Advances in neural information processing systems*, 2016, pp. 3675–3683.

[11] M. Garnelo, K. Arulkumaran, and M. Shanahan, "Towards deep symbolic reinforcement learning," *arXiv preprint arXiv:1609.05518*, 2016.

[12] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. R. Salakhutdinov, and A. J. Smola, "Deep sets," in *Advances in Neural Information Processing Systems*, 2017, pp. 3394–3404.

[13] A. Santoro, D. Raposo, D. G. Barrett, M. Malinowski, R. Pascanu, P. Battaglia, and T. Lillicrap, "A simple neural network module for relational reasoning," in *Advances in neural information processing systems*, 2017, pp. 4974–4983.

[14] N. Watters, A. Tacchetti, T. Weber, R. Pascanu, P. Battaglia, and D. Zoran, "Visual interaction networks," *arXiv preprint arXiv:1706.01433*, 2017.

[15] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International Conference on Machine Learning*, 2016, pp. 1928–1937.

[16] A. Pritzel, B. Uria, S. Srinivasan, A. Puigdomènech, O. Vinyals, D. Hassabis, D. Wierstra, and C. Blundell, "Neural episodic control," *arXiv preprint arXiv:1703.01988*, 2017.

[17] C. Tessler, S. Givony, T. Zahavy, D. J. Mankowitz, and S. Mannor, "A deep hierarchical approach to lifelong learning in minecraft." in *AAAI*, vol. 3, 2017, p. 6.

[18] C. Diuk, A. Cohen, and M. L. Littman, "An object-oriented representation for efficient reinforcement learning," in *Proceedings of the 25th international conference on Machine learning*. ACM, 2008, pp. 240–247.

[19] D. E. Hershkowitz, J. MacGlashan, and S. Tellex, "Learning propositional functions for planning and reinforcement learning," in *2015 AAAI Fall Symposium Series*, 2015.

[20] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," *arXiv preprint arXiv:1612.00593*, 2016.

[21] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[22] D. Perez-Liebana, J. Liu, A. Khalifa, R. D. Gaina, J. Togelius, and S. M. Lucas, "General video game ai: a multi-track framework for evaluating agents, games and content generation algorithms," *arXiv preprint arXiv:1802.10363*, 2018.

[23] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *Journal of Artificial Intelligence Research*, 2012.

[24] T. Schaul, "A video game description language for model-based or interactive learning," in *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*. IEEE, 2013, pp. 1–8.

[25] Y. Liang, M. C. Machado, E. Talvitie, and M. Bowling, "State of the art control of atari games using shallow reinforcement learning," *arXiv preprint arXiv:1512.01563*, 2015.

Automated Curriculum Learning by Rewarding Temporally Rare Events

Niels Justesen
IT University of Copenhagen
Copenhagen, Denmark
noju@itu.dk

Sebastian Risi
IT University of Copenhagen
Copenhagen, Denmark
sebr@itu.dk

***Abstract*—Reward shaping allows reinforcement learning (RL) agents to accelerate learning by receiving additional reward signals. However, these signals can be difficult to design manually, especially for complex RL tasks. We propose a simple and general approach that determines the reward of pre-defined events by their rarity alone. Here events become less rewarding as they are experienced more often, which encourages the agent to continually explore new types of events as it learns. The adaptiveness of this reward function results in a form of automated curriculum learning that does not have to be specified by the experimenter. We demonstrate that this *Rarity of Events* (RoE) approach enables the agent to succeed in challenging VizDoom scenarios without access to the extrinsic reward from the environment. Furthermore, the results demonstrate that RoE learns a more versatile policy that adapts well to critical changes in the environment. Rewarding events based on their rarity could help in many unsolved RL environments that are characterized by sparse extrinsic rewards but a plethora of known event types.**

***Index Terms*—reinforcement learning, intrinsic motivation, VizDoom, reward shaping**

I. Introduction

Deep reinforcement learning and deep neuroevolution have achieved impressive results learning to play video games [17] and controlling both simulated and physical robots [2, 9, 15, 27]. These approaches, however, struggle to learn in environments where feedback signals (also called rewards) are sparse and/or delayed. A popular way to overcome this issue is to shape the reward function with prior knowledge such that the agent receives additional rewards to guide its learning process [22, 23, 31]. Another approach is to gradually increase the difficulty of the environment to ease learning through curriculum learning [4, 46]. Both approaches are time-consuming, require substantial domain knowledge and are especially difficult to implement for complex environments. In this paper, we propose a simple method that automatically shapes the reward function during training and performs a form of curriculum learning that adapts to the agent's current performance. The only required domain knowledge is the specification of a set of positive events that can happen in the environment (e.g. picking up items, moving, winning etc.), which is easy to implement if raw state changes are accessible.

The method introduced in this paper rewards a reinforcement learning (RL) agent by the rarity of experienced events such that rare events have a higher value than frequent events. The idea is to completely discard the extrinsic reward and instead motivate the agent intrinsically toward a behavior that explores the pre-defined events. As the agent first experiences certain types of events that are relatively easy to learn (e.g. moving around and picking up items) they will slowly become less rewarding, pushing the agent to explore rare and potentially more difficult events. Thus by only rewarding events for their rarity, the system performs a form of automated curriculum learning.

The goal of this approach is to learn through a process of *curiosity* rather than optimizing toward a difficult pre-defined goal. We apply our method, called *Rarity of Events* (RoE), to learn agent behaviors from raw pixels in the VizDoom framework [19]. While our approach could be applied to any reward-based learning method and possibly also fitness-based evolutionary methods, in this paper we train deep convolutional networks through the actor-critic algorithm A2C [29]. In the future, RoE could offer a new way to learn versatile behaviors in increasingly complex environments such as StarCraft, which is a yet unsolved reinforcement learning problem [44].

The paper is structured as follows. We first review relevant previous work, including related approaches in Section II. After explaining RoE (Section III), we demonstrate the usefulness of the method on five challenging VizDoom scenarios with sparse rewards and show how RoE learns a versatile behavior that can adapt to critical changes in the environment (Section V).

II. Previous Work

A. Deep Reinforcement Learning

Deep reinforcement learning allows learning agent behaviors in video games directly from screen pixels, including Atari games [28], first-person shooters [19, 22, 46], and car racing games [29]. These methods are typically variants of Deep Q Networks (DQN) [28] or actor-critic methods with parallel actor-learners such as Asynchronous Advantage Actor-Critic (A3C) [29]. Neuroevolution [13, 36] has also recently shown promising results in playing Atari games and can be easier to parallelize [39, 43].

A key requirement for deep RL methods to work out of the box are frequent and easy obtainable reward signals from the environment that can guide learning toward an optimal

978-1-5386-4359-4/18/$31.00 ©2018 IEEE

behavior. An infamous Atari game where this is not the case is Montezuma's Revenge; for this game with very sparse rewards, both DQN and A3C variants fail [28, 29].

The lack of frequent reward signals can be overcome by reward shaping, where a smoother reward function is designed using prior domain knowledge [22, 23], or by gradually increasing the difficulty of the environment (e.g. the level itself or the NPCs' behaviors) to ease learning through curriculum learning [4, 46]. Related to curriculum learning is a method called *Power Play* that searches for new unsolvable problems while the agent is trained to progressively match the difficulty of the environment [41]. Another related approach is hierarchical reinforcement learning where a meta-controller controls one or more sub-policies that are trained to reach sub-goals (equivalent to events) [7, 20].

B. Curiosity & Intrinsic Motivation

In curiosity-driven learning the agent seeks to explore new situations guided by intrinsic motivation [18, 32, 38]. One theory of intrinsic motivation is *reduction of cognitive dissonance*, i.e. the motivation to learn a cognitive model that can explain and predict sensory input [12, 33]. This theory has also been formalized in the context of RL in which agents are intrinsically rewarded when observing temporarily novel, interesting, or surprising patterns based on their own world model [40]. A related idea is *optimal incongruity*, where discrepancies between the currently perceived and what is usually perceived produce a high stimulus; thus novel situations that yet lie within our current understanding are highly rewarding [5, 16]. The prediction error of a learning model can thus be used directly to define the reward function [14].

One way of implementing intrinsic motivation is to model the expected learning progress $\zeta(s, a)$ of a state-action pair [25]. The Intrinsic Curiosity Module (ICM) is another approach that encodes states s_t and s_{t+1} into features $\Phi(s_t)$ and $\Phi(s_{t+1})$ and determines the intrinsic reward based on the prediction error of these features and the forward model's features [34]. State-density models that assign probabilities to screen images, can be learned together with a policy and then determine intrinsic motivation as the model's temporal change in prediction, such that surprising screen images produce higher rewards [3].

Rewarding RL agents based on the novelty of events has been explored earlier with tabular Q-learning in a simple 3D environment [26], where the reward is highest when novelty is moderate. A combination of habituation theory and self-organizing maps was employed to vary the agent's curiosity (the reward signal toward certain events).

C. Novelty Search

The pursuit of novel situations also shares some similarities with novelty search [24] in evolutionary computation. The idea of novelty search is to search for novel behaviors instead of optimizing toward a specific objective directly. Both novelty search and our approach RoE push the search toward unexplored areas; however, novelty search does so for a population of individuals where novelty is defined as the behavioral distance to other behaviors in the population. Our approach is trained through reinforcement learning and novelty (or rather rarity of events) is based on experiences of previous versions of the policy.

D. VizDoom

The approach in this paper is tested in VizDoom, an AI research platform based on the commercial video game Doom that allows learning from raw visual information [19]. The VizDoom framework includes several diverse environments, some of which are very challenging to learn due to their sparse and delayed rewards. Several deep RL approaches have been applied to Doom, which include auxiliary learning [21, 22], game-feature augmentation [6, 8, 11], manual reward shaping [8, 11, 22], and curriculum learning [46]. A very different approach by Alvernaz and Togelius applies neuroevolution on top of a pre-trained auto-encoder [1]. In this paper, we purposefully build on a vanilla implementation of the RL algorithm A2C, to set a baseline for how well RoE can help in challenging VizDoom scenarios.

III. Approach

This section describes our *Rarity of Events* (RoE) approach and its integration with A2C in VizDoom.

A. Rewarding Temporally Rare Events

The reward function in RoE adapts throughout training to the policy's ability to explore the environment. By rewarding events based on how often they occur during training, the agent is intrinsically motivated toward exploring new parts of the environment rather than aiming for a single goal that might be difficult to obtain directly. In effect, the approach performs a form of curriculum learning since events are rewarded based on the agent's current ability to obtain them. As the agent learns, it becomes less interested in events that are frequent and *curious* about newly discovered events.

Our method requires a set of pre-defined events, and the reward $R_t(\epsilon_i)$ for experiencing one of these events ϵ_i at time t is determined by its temporal rarity $\frac{1}{\mu_t(\epsilon_i)}$, where $\mu_t(\epsilon_i)$ is the temporal episodic mean occurrence of ϵ_i at time t, i.e. how often ϵ_i occurs per episode at the moment. The mean occurrences of events are clipped to be above a lower threshold τ (we used 0.01 such that the maximum reward for any event is 100). For a vector of event occurrences x, such that x_i is the number of times ϵ_i occurred in a game step, the reward is the sum of all event rewards:

$$R_t(x) = \sum_{i=1}^{|x|} x_i \frac{1}{\max(\mu_t(\epsilon_i), \tau)}. \quad (1)$$

The rarity measure $\frac{1}{\mu_t(\epsilon_i)}$ is not arbitrary but is designed such that all events have equal importance. If any event ϵ_i is experienced n times during an episode, and $n = \mu_t(\epsilon_i)$ (which is the expected amount), then the accumulated reward for ϵ_i is 1 regardless of the rarity. This means that in theory all events have equal importance. In practice, the policy might

learn that some events have a negative or positive influence on the occurrence of others.

B. Determining the Temporal Episodic Mean Occurrence

There are arguably many ways to determine the temporal episodic mean occurrence $\mu_t(\epsilon_i)$; here we employ a simple approach that nevertheless achieves the desired outcome. Whenever an episode during training reaches a terminal state, a vector ϵ containing the occurrence of events in this episode is added to a buffer of size N. The size of the buffer determines the adaptability of the reward function. If N is small, the agent quickly becomes *bored* of new events as it easily forgets their rarity in the past. If N is large, the agent will stay *curious* for a longer period of time. The temporal episodic mean occurrence $\mu_t(\epsilon)$ is then determined as the mean of all records in the buffer, i.e. the episodic mean of the last N episodes.

C. Events in Doom

We track 26 event types in VizDoom by implementing a function that determines which events occur in every state transition (i.e. in each time step). The event types include movement (one unit), shooting (decrease in ammo), picking up an item (one event for each item type; health pack, armor, ammo, and weapons 0–9), killing (one for each weapon type 0–9 as well as one regardless of weapon type). Movement events are triggered when the agent has traveled one unit from the position of the last movement event (or the initial position if the agent has not yet moved).

D. Policy

The presented reward shaping approach can be applied to most (if not all) RL methods that learn from a reward signal. It could potentially also be applied to evolutionary approaches such as Evolution Strategies by defining fitness as the sum of rewards in an episode. A standard policy network is employed that has three convolutional layers followed by a fully connected layer of 512 units, and a policy and value output. We use filter sizes of [32, 64, 32] with strides [4, 2, 1], ReLU activations for hidden layers, and softmax for the policy output.

The input is a single frame of 160×120 pixels in grayscale, cropped by removing 10 pixels on top/bottom and 30 pixels on the sides and then resized to 80×80. In most of the scenarios, the agent can perform four actions: attack, move forward, turn left, and turn right. In this case, the policy output has $2^4 = 16$ values to allow any combination of the four actions. The event buffer is updated whenever a worker reaches a terminal state. The rewards from VizDoom, which vary between -100 and 100, are normalized to $[0, 1]$. Rewards based on our approach are not normalized and vary between $[0, 100]$ (due to $\tau = 0.01$), while for all events where $\mu_t(\epsilon_i) \geq 1$ the reward will be between 0 and 1 (following Equation 1 in Section III-A).

E. Advantage Actor-Critic (A2C)

The deep networks in this paper are trained with the deep reinforcement learning algorithm A2C, a synchronous variant of Asynchronous Advantage Actor-Critic (A3C) [29], which is able to reach state-of-the-art performance in a wide range of environments [42, 45, 47].

A2C is an actor-critic method that optimizes both a policy π (the actor) and an estimation of the state-value function $V(s)$ (the critic). Parallel worker threads share the same model parameters and synchronously collect trajectories $(s_t, s_{t+1}, a_t, r_{t+1})$ for t_{max} game steps where after the model's parameters are updated. Threads restart new episodes individually when they are done. The discounted return $R_t = \sum_{i=1}^{k-1} \gamma^i r_{t+i} + \gamma^k V(s_{t+k})$, where k is the number of trajectories collected after t, and the advantage $A(s_t, a_t) = R_t - V(s_t)$ is determined for each step, for every worker. A2C then uses the traditional A3C update rules in [29] based on the policy loss $\log \pi(a_i|s_i)A(s_i)$ and value loss; the mean squared error between the experienced R_t and the predicted $V(s_t)$: $\frac{1}{2}(R_t - V(s_t))^2$. In contrast to A3C, A2C updates the parameters synchronously in batches.

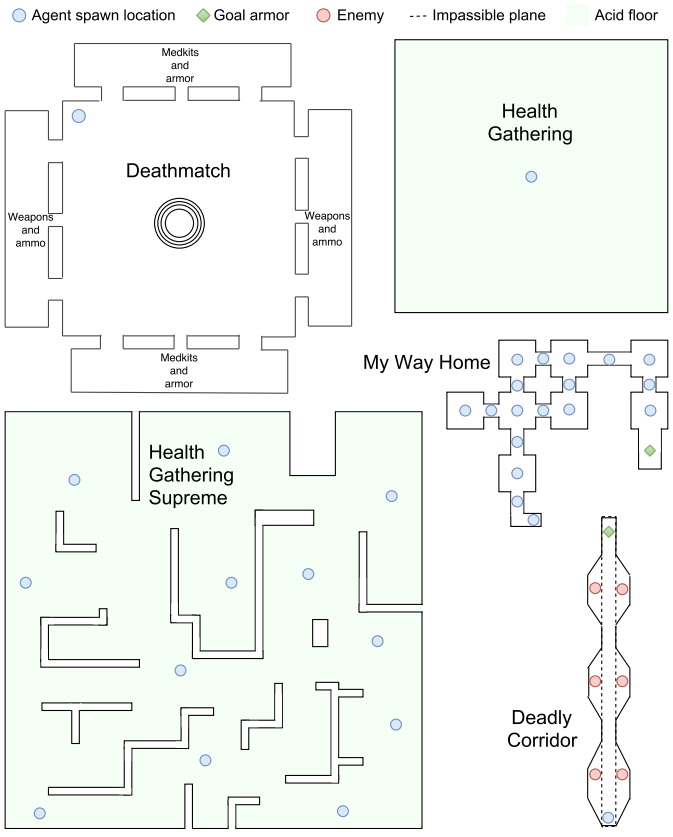


Fig. 1: The five ViZdoom scenarios. Scenarios with multiple spawning positions randomly select one of them at the start of an episode. The episode ends when the goal armor, which only appears in *My Way Home* and *Deadly Corridor*, is picked up. The agent periodically looses health when standing on acid floors.

IV. VizDoom Testing Scenarios

This section describes the five VizDoom scenarios used in our experiments. They all have sparse and/or delayed rewards

and are therefore a good test domain for our approach. The scenarios are from the original VizDoom [19] repository[1].

For each scenario we also detail the extrinsic reward from the environment, which is used when training models without RoE. Some of these extrinsic rewards were rescaled to be coherent across scenarios. If not stated otherwise, the agent can move forward, turn left, turn right, and shoot. Screenshots from these scenarios are shown in Figure 2, with top-down views in Figure 1.

1) Health Gathering: The goal is to survive as long as possible in a square room with an acid floor that deals damage periodically. Medkits spawn randomly in the room and can help the agent to survive as they heal when picked up. The agent is rewarded 1 for every time step it is alive, and -100 for dying. The maximum episode length is 2,100 time steps. The agent cannot shoot.

2) Health Gathering Supreme: Same as *Health Gathering* but within a maze.

3) My Way Home: The goal is to pick up an armor, which gives a reward of 100 and ends the scenario immediately. The agent cannot shoot and is rewarded -0.1 for every time step it is alive. The agent starts an episode at one of the randomly chosen spawn locations with a random rotation.

4) Deadly Corridor: Similarly to *My Way Home*, the goal is to pick up an armor, which gives a reward of 100 and ends the scenario immediately. The armor is located at the end of a corridor, which is guarded by enemies on both sides. The agent must kill most, if not all of the enemies to reach it, and receives a -100 reward if it dies. The original reward shaping function (the distance to the armor) has been removed to make it harder and to compare RoE with a baselines that does not use any reward shaping. The maximum episode length is 2,100 time steps.

Fig. 2: From top-left to bottom-right: Screenshot from *Deathmatch*, *My Way Home*, *Health Gathering Supreme*, and *Deadly Corridor*. Notice that in some scenarios the agent cannot shoot. The scenario *Health Gathering* is similar to *Health Gathering Supreme* but without walls within the room.

5) Deathmatch: The agent spawns in a large battle arena with an open area in the middle and four rooms, one in each direction that contain either medkits and armor, or weapons (chainsaw, super shotgun, chaingun, rocket launcher, and plasma gun) and ammunition for each weapon. The maximum episode length is 4,200 time steps. The agent is rewarded the following amounts when killing an enemy: Zombieman (100), ShotgunGuy (300), MarineChainsawVzd (300), Demon (300), ChaingunGuy (400), HellKnight (1,000). These enemies spawn randomly on the map when the scenario starts.

To test how well the approach can adapt to new scenarios, five variations of *Deathmatch* were also created that only include a certain weapon type. These scenarios are called *Deathmatch Chainsaw*, *Deathmatch Chaingun*, *Deathmatch Shotgun*, *Deathmatch Plasma*, and *Deathmatch Rocket* to denote which weapon that remains on the map. The ammunition for the other weapons was also removed.

A2C	
Learning rate	7e-4
γ (discount factor)	0.99
Entropy coefficient	0.01
Value loss coefficient	0.5
Learning rate	0.0007
Max. gradient-norm	0.5
Worker threads	4 (16 in DM)
t_{max} (Steps per. update)	20
Batch size	64
Frame skip	4
RMSprop Optimizer	
ϵ	1e-5
α	0.99
RoE	
N (event buffer size)	100
τ (mean threshold)	0.01

TABLE I: Experimental configurations for A2C and A2C+RoE. 16 worker threads were used in *Deathmatch*.

V. Results

We tested A2C with our approach *Rarity of Events* (A2C+RoE) on the five VizDoom scenarios described in Section IV. The *Deathmatch* variations were not used for training. As a comparison baseline, A2C was also trained using the extrinsic reward from the environment as described in Section IV. Due to computational constraints we only trained each method once on each scenario.

When training with A2C+RoE, the agent did not have access to the extrinsic reward throughout training but only the intrinsic reward based on the temporal rarity of the pre-defined events. The algorithms ran for 10^7 time steps for each scenario and 7.5×10^7 for the Deathmatch scenario. For both A2C and A2C+RoE we save a copy of the model parameters whenever the mean extrinsic reward across all workers improves. The last copy is considered to be the final model that we use in our tests. The complete configurations for A2C and A2C+RoE are shown in Table I and the code for the experiments and trained models are available on GitHub[2]. Videos of the learned policies are available on YouTube[3].

[1]https://github.com/mwydmuch/ViZDoom/tree/master/scenarios

[2]https://github.com/njustesen/rarity-of-events

[3]https://youtu.be/YG-lf732a0U

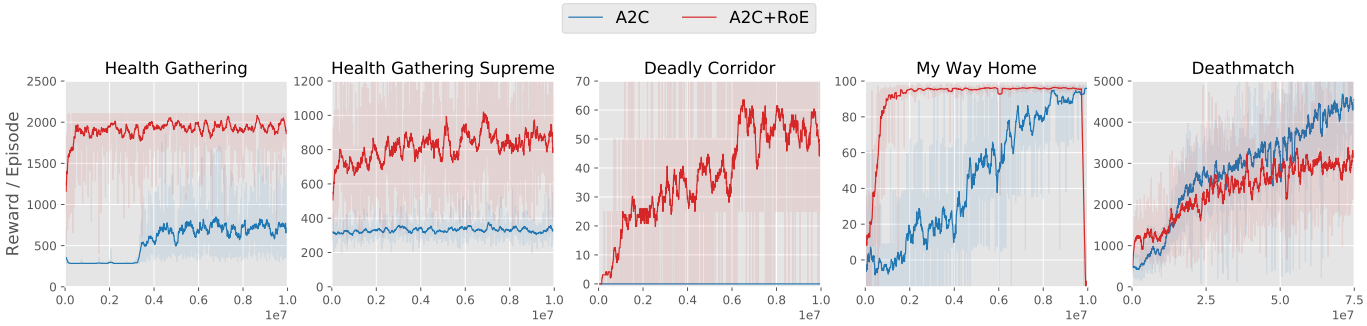


Fig. 3: The reward per episode of A2C and A2C+RoE during training in five VizDoom scenarios (smoothed). A2C is trained from the environment's extrinsic reward while A2C+RoE uses our proposed method without access to the reward. The drop in performance seen in the My Way Home scenario is discussed in-depth in Section V-A.

Fig. 4: Episodic mean occurrence during training for a subset of the event types in the five VizDoom scenarios. Notice the last spike in the *My Way Home* scenario with A2C+RoE, in which the policy ignores the final goal (armor pickup) to prioritize continuous movement around the maze.

Scenario	A2C	A2C+RoE	t-test
Health Gathering	399 ± 107	**1261** ± 533	$p < 0.0001$
Health Gathering Supr.	305 ± 60	**1427** ± 645	$p < 0.0001$
Deadly Corridor	0.00 ± 0.0	**40** ± 49	$p < 0.0001$
My Way Home	96.69 ± 0.12	**97.89** ± 0.01	$p < 0.0001$
Deathmatch	**4611** ± 2595	4062 ± 2442	$p = 0.1250$
Deathmatch Chainsaw	1025 ± 809	**3750** ± 3130	$p < 0.0001$
Deathmatch Chaingun	1487 ± 1189	**2852** ± 2038	$p < 0.0001$
Deathmatch Shotgun	1375 ± 941	**1832** ± 1752	$p = 0.0226$
Deathmatch Plasma	**4538** ± 1537	3248 ± 2701	$p < 0.0001$
Deathmatch Rocket	616 ± 583	**1463** ± 1449	$p < 0.0001$

TABLE II: Shown are average scores based on evaluating the best policies found for A2C and A2C+RoE 100 times each. The best results are shown in bold. The five last rows show how the policies that were trained on the original *Deathmatch* scenario generalize to five variations where only one weapon type is available. Standard deviations are shown for each experiment and two-tailed p-values from unpaired t-tests.

A. Learned Policies

The A2C baseline did not learn a good policy in *Health Gathering Supreme* and *Deadly Corridor*, and only improved slightly in *Health Gathering* (Figure 3). A2C learned a weak policy in three out of five scenarios, which demonstrates that they are indeed difficult to master guided by the extrinsic rewards alone. In *My Way Home*, A2C does learn a strong behavior that consistently locates and picks up the armor but only after 8–9 million training steps. In *Deathmatch*, A2C learned a very high-performing behavior that directly walks to the plasma gun (the most powerful weapon in this scenario) and shoots from cover toward the center of the map. The behavior is simple but effective until it runs out of ammunition, after which it attempts to find more ammunition and sometimes fails.

Our approach A2C+RoE learns effective behaviors in all five scenarios. The learned behavior in *Deathmatch* does not exclusively use the powerful plasma gun, which results in a slightly but not significantly worse performance than A2C ($p = 0.125$ using two-tailed t-test). The policy is still effective with over 10 kills per episode. These kills are spread across all weapons that are available, resulting in a behavior that is more varied (and interesting to watch). As we will show in Section V-B, the versatile behavior learned by A2C+RoE allows it to adapt to critical changes in *Deathmatch* in contrast

to policies trained through A2C.

The episodic mean occurrence of events (Figure 4) allows us to analyze how the policies change over time. In *Health Gathering* and *Health Gathering Supreme*, A2C+RoE quickly learns to move $\sim$80 and $\sim$30 units per episode, respectively. This behavior might explain why the agent also quickly learns to pick up medkits. A2C, on the other hand, learns the relationship between movement, medkits, and survival at a much slower pace, at least in the *Health Gathering* scenario. In *Deadly Corridor* A2C+RoE discovers an interesting behavior. After the agent learns to kill all six enemies (the red line) and to pick up armor (purple line), it still manages to increase the movement and the shooting events; the agent learned to walk back to its initial position while shooting and then afterwards to return to pick up the armor. This result is not unexpected as the agent is intrinsically motivated to experience as many events as possible during an episode.

In *My Way Home*, after the A2C+RoE policy has learned to routinely pick up the armor, it shifts into a different behavior toward the end of training. The agent learned to avoid the armor to instead continuously move around in the maze. We suspect that the policy would shift back to the previous behavior if training was continued, as the movement reward is now decreasing and the armor reward is increasing. Since our rarity measure is temporal, loops between these two behaviors could emerge as well. As policies with the highest extrinsic reward are saved during training, these sudden changes do not affect the final policy. In fact, one might argue that this is a useful feature of RoE: a network that has converged to some optimum can escape it to find other interesting behaviors.

B. Ability to Adapt

A2C+RoE motivates the agent intrinsically to learn a balanced policy that strives to experience a good mix of events. Reinforcement learning algorithms that exclude pre-training or proper reward shaping, including our A2C baseline, can easily converge into local optima with very *narrow* behaviors. In this context, *narrow* refers to behaviors that act in a very particular way, only utilizing a small subset of the features in the environment. This handicap prevents the learned policies from adapting to critical changes in the environment as they only know one way of behaving.

To test for such adaptivity, the learned policies are evaluated on five *Deathmatch* variations in which critical weapons and ammunition packs have been removed. Note that the policies were not directly trained on these variations. The results in Table II show that A2C+RoE learned a policy that significantly outperforms A2C ($p < 0.0001$ using two-tailed t-test) in four out of five *Deathmatch* variations. A2C+RoE learned a policy that is more versatile, capable of using all the weapons in the map, which is the reason it can easily adapt. Figure 5 shows heat maps (i.e. the proportional time spent at each map location) during the evaluations of the two policies on *Deathmatch* and its variations. The A2C+RoE policy expresses different strategies depending on the weapon available on the map, while the A2C policy mostly circles around the plasma gun location, regardless of it actually being there. However, if the plasma gun is present, A2C alone does execute a fairly effective strategy, shooting toward enemies in the middle of the map.

The heat maps show that the A2C policy has learned to stay at only one location on the map from which it can pick up the powerful plasma gun and thereafter shoot efficiently toward enemies in the middle of the map (see the video of the learned policies). In the *Deathmatch* variations, in which the map only contains two weapons of the same type, the A2C-policy fails to adapt to use the other weapons and instead walks around the area where the plasma gun would have been located.

The A2C+RoE policy has learned to explore a larger part of the maps in a more uniform way (Figure 5,bottom). In the different *Deathmatch* variations, a clear change in behavior can be observed when only a certain type of weapon is available. For example, in the *DM Rocket* scenario, the agent lures enemies into the map's top and bottom room while efficiently using the rocket's splash damage.

VI. Discussion

While the presented approach worked well in VizDoom it will be important to test its generality in other domains in the future. RoE is designed to work well in challenging environments that have a plethora of known events and sparse and/or delayed rewards. Video games are thus a very suitable domain and we plan to test RoE in Montezuma's Revenge and StarCraft in future work. For domains in which reward shaping is not necessary, i.e. the extrinsic reward smoothly leads to an optimal behavior, our approach might add less value. We imagine that RoE should also work well in domains with deceptive reward structures, just as novelty search outperformed traditional evolutionary algorithms in mazes with dead ends [24] or deceptive meta-learning tasks [37]. Novelty search and RoE have the ability to learn interesting behaviors without the need for a goal. In the future, our approach could also be extended to reward the agent for both the rarity of events as well as the environment's original objective, inspired by *quality diversity methods* [35] that use a combination of diversity and objective-based search [10, 30].

The specification of adequate events is intimately tied to the success of our approach; events that lead to direct negative performance should be avoided. For example, if the extrinsic reward is negative when the agent wastes ammunition, it should not be intrinsically rewarded for shooting event. A benefit of the presented method is that events that contribute to the occurrence of other events (e.g. such as movement leads to medkit pickups), can lead to a system that performs automated curriculum learning. However, it is not guaranteed that this effect will occur, and it might require a bit of trial and error during the specification of events. Some events can also be contradicting, such as killing with the chainsaw and killing with the plasma gun, as the agent cannot do both at the same time. Our approach is designed to learn a policy that can balance their occurrences which results in a more versatile

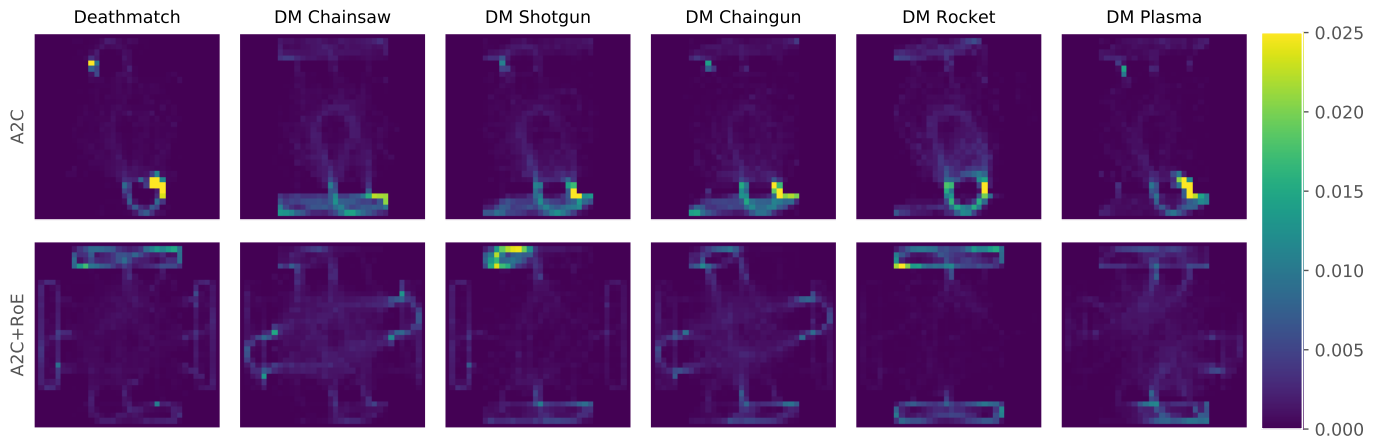


Fig. 5: Heat maps showing the proportional time spent at each location on the map in the *Deathmatch* scenario and its five variations. The values are based on evaluating the two trained policies 100 times each and clipped at 0.025. The heat maps show that the A2C-policy prefers to stay near the plasma gun, even in the map variations where it is not present, while the A2C+RoE-policy has learned distinct behaviors for each weapon type. The results in Table II shows that the A2C+RoE-policy is able to reach high scores in these variations event though it was never trained on them.

behavior. Important future work will test how RoE scales to hundreds or even thousands of events. A promising testbed for such experimentation is StarCraft, for which events can easily be defined as the production of each unit and building type, as well as killing different opposing unit types. We believe that reinforcement learning methods that are guided by intrinsic motivation are key to solving these challenging environments.

The A2C baseline reached the best performance in the original *Deathmatch*. However, it can be argued whether it learned to actually play Doom, or just learned to follow a fixed sequence of actions that lead to the same behavior every time. While it can be useful to find a niche behavior with high performance, learning a rich and versatile behavior has particular relevance for video games. Here, behaviors that explore the game's features could potentially help for automatic game testing and also lead to more human-like behaviors for NPCs.

Regarding our implementation of the RoE approach, future work will also explore other variations in determining the episodic mean occurrence of events, such as discounting the mean occurrences over time. With this modification, event occurrences older than N episodes (the event buffer only holds N event occurrences) would still effect the intrinsic reward.

It is important to note that since we save the best model based on the mean extrinsic reward across all worker threads, increasing the number of threads should make the evaluation less noisy by reducing the chances of accidentally overriding the best model with a worse performing one. This hypothesis still needs to be confirmed, but the number of threads was already increased from 4 to 16 in the longer *Deathmatch* scenario to speed up learning.

VII. Conclusion

We introduced *Rarity of Events* (RoE), a simple reinforcement learning approach that determines reward based on the temporal rarity of pre-defined events. This approach was able to reach high-performing scores in five challenging VizDoom scenarios with sparse and/or delayed rewards. Compared to a traditional A2C baseline, the results are significantly better in four of the five scenarios. Importantly, the presented approach is able to not only receive a high final reward, but also discovers versatile behavior that can adapt to critical changes in the environment, which is challenging for the baseline A2C approach. In our experiments, the extrinsically motivated baseline either fails in these environments or learns a behavior that is unable to adapt to changes in the environments it has been trained on. In the future, the presented RoE approach could allow more complex scenarios to be solved, for which it is infeasible to learn from extrinsic rewards without manual reward shaping and curriculum learning.

VIII. Acknowledgements

We thank OpenAI for publishing accessible implementations of A2C and GitHub user p-kar for the integration of A2C to VizDoom[4]. We would also like to show our gratitude to the members of the Game Innovation Lab at New York University Tandon School of Engineering for their feedback and inspiring ideas. This work was financially supported by the Elite Research travel grant from The Danish Ministry for Higher Education and Science.

References

[1] S. Alvernaz and J. Togelius. Autoencoder-augmented neuroevolution for visual doom playing. In *Computational Intelligence and Games (CIG), 2017 IEEE Conference on*, pages 1–8. IEEE, 2017.

[2] M. Andrychowicz, D. Crow, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. P. Abbeel, and W. Zaremba. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, pages 5055–5065, 2017.

[3] M. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos. Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information Processing Systems*, pages 1471–1479, 2016.

[4]https://github.com/p-kar/a2c-acktr-vizdoom

[4] Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48. ACM, 2009.

[5] D. E. Berlyne. Conflict, arousal, and curiosity. 1960.

[6] S. Bhatti, A. Desmaison, O. Miksik, N. Nardelli, N. Siddharth, and P. H. Torr. Playing doom with slam-augmented deep reinforcement learning. *arXiv preprint arXiv:1612.00380*, 2016.

[7] M. M. Botvinick, Y. Niv, and A. C. Barto. Hierarchically organized behavior and its neural foundations: a reinforcement learning perspective. *Cognition*, 113(3):262–280, 2009.

[8] D. S. Chaplot and G. Lample. Arnold: An autonomous agent to play fps games. In *AAAI*, pages 5085–5086, 2017.

[9] Y. Chebotar, K. Hausman, M. Zhang, G. Sukhatme, S. Schaal, and S. Levine. Combining model-based and model-free updates for trajectory-centric reinforcement learning. *arXiv preprint arXiv:1703.03078*, 2017.

[10] G. Cuccu and F. Gomez. When novelty is not enough. In *European Conference on the Applications of Evolutionary Computation*, pages 234–243. Springer, 2011.

[11] A. Dosovitskiy and V. Koltun. Learning to act by predicting the future. *arXiv preprint arXiv:1611.01779*, 2016.

[12] L. Festinger. *A Theory of Cognitive Dissonance*. Stanford University Press, 1957. ISBN 9780804709118.

[13] D. Floreano, P. Dürr, and C. Mattiussi. Neuroevolution: from architectures to learning. *Evolutionary Intelligence*, 1(1):47–62, 2008.

[14] G. Gordon and E. Ahissar. Hierarchical curiosity loops and active sensing. *Neural Networks*, 32:119–129, 2012.

[15] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine. Continuous deep Q-learning with model-based acceleration. In *International Conference on Machine Learning*, pages 2829–2838, 2016.

[16] J. M. HUNT. Intrinsic motivation and its role in psychological development. *Nebraska symposium on motivation*, 13:189–282, 1965. URL https://ci.nii.ac.jp/naid/20001159493/en/.

[17] N. Justesen, P. Bontrager, J. Togelius, and S. Risi. Deep learning for video game playing. *arXiv preprint arXiv:1708.07902*, 2017.

[18] F. Kaplan and P.-Y. Oudeyer. Intrinsically motivated machines. In *50 years of artificial intelligence*, pages 303–314. Springer, 2007.

[19] M. Kempka, M. Wydmuch, G. Runc, J. Toczek, and W. Jaśkowski. Vizdoom: A doom-based ai research platform for visual reinforcement learning. In *Computational Intelligence and Games (CIG), 2016 IEEE Conference on*, pages 1–8. IEEE, 2016.

[20] T. D. Kulkarni, K. Narasimhan, A. Saeedi, and J. Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Advances in neural information processing systems*, pages 3675–3683, 2016.

[21] T. D. Kulkarni, A. Saeedi, S. Gautam, and S. J. Gershman. Deep successor reinforcement learning. *arXiv preprint arXiv:1606.02396*, 2016.

[22] G. Lample and D. S. Chaplot. Playing FPS games with deep reinforcement learning. In *AAAI*, pages 2140–2146, 2017.

[23] A. D. Laud. Theory and application of reward shaping in reinforcement learning. Technical report, 2004.

[24] J. Lehman and K. O. Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary computation*, 19(2):189–223, 2011.

[25] M. Lopes, T. Lang, M. Toussaint, and P.-Y. Oudeyer. Exploration in model-based reinforcement learning by empirically estimating learning progress. In *Advances in Neural Information Processing Systems*, pages 206–214, 2012.

[26] M. L. Maher, K. E. Merrick, and R. Saunders. Achieving creative behavior using curious learning agents. In *AAAI spring symposium: Creative intelligent systems*, volume 8, pages 40–46, 2008.

[27] P. Mirowski, R. Pascanu, F. Viola, H. Soyer, A. J. Ballard, A. Banino, M. Denil, R. Goroshin, L. Sifre, K. Kavukcuoglu, et al. Learning to navigate in complex environments. *arXiv preprint arXiv:1611.03673*, 2016.

[28] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[29] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937, 2016.

[30] J.-B. Mouret and J. Clune. Illuminating search spaces by mapping elites. *arXiv preprint arXiv:1504.04909*, 2015.

[31] A. Y. Ng. *Shaping and policy search in reinforcement learning*. PhD thesis, University of California, Berkeley, 2003.

[32] P.-Y. Oudeyer. Computational theories of curiosity-driven learning. *arXiv preprint arXiv:1802.10546*, 2018.

[33] P.-Y. Oudeyer and F. Kaplan. What is intrinsic motivation? a typology of computational approaches. *Frontiers in Neurorobotics*, 1:6, 2009.

[34] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell. Curiosity-driven exploration by self-supervised prediction. In *International Conference on Machine Learning (ICML)*, volume 2017, 2017.

[35] J. K. Pugh, L. B. Soros, and K. O. Stanley. Quality diversity: A new frontier for evolutionary computation. *Frontiers in Robotics and AI*, 3:40, 2016.

[36] S. Risi and J. Togelius. Neuroevolution in games: State of the art and open challenges. *IEEE Transactions on Computational Intelligence and AI in Games*, 9(1):25–41, 2017.

[37] S. Risi, C. E. Hughes, and K. O. Stanley. Evolving plastic neural networks with novelty search. *Adaptive Behavior*, 18(6): 470–491, 2010.

[38] R. M. Ryan and E. L. Deci. Intrinsic and extrinsic motivations: Classic definitions and new directions. *Contemporary educational psychology*, 25(1):54–67, 2000.

[39] T. Salimans, J. Ho, X. Chen, and I. Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*, 2017.

[40] J. Schmidhuber. Formal theory of creativity, fun, and intrinsic motivation (1990–2010). *IEEE Transactions on Autonomous Mental Development*, 2(3):230–247, 2010.

[41] J. Schmidhuber. Powerplay: Training an increasingly general problem solver by continually searching for the simplest still unsolvable problem. *Frontiers in psychology*, 4:313, 2013.

[42] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[43] F. P. Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley, and J. Clune. Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. *arXiv preprint arXiv:1712.06567*, 2017.

[44] O. Vinyals, T. Ewalds, S. Bartunov, P. Georgiev, A. S. Vezhnevets, M. Yeo, A. Makhzani, H. Küttler, J. Agapiou, J. Schrittwieser, et al. Starcraft II: a new challenge for reinforcement learning. *arXiv preprint arXiv:1708.04782*, 2017.

[45] J. X. Wang, Z. Kurth-Nelson, D. Tirumala, H. Soyer, J. Z. Leibo, R. Munos, C. Blundell, D. Kumaran, and M. Botvinick. Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763*, 2016.

[46] Y. Wu and Y. Tian. Training agent for first-person shooter game with actor-critic curriculum learning. 2016.

[47] Y. Wu, E. Mansimov, R. B. Grosse, S. Liao, and J. Ba. Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. In *Advances in neural information processing systems*, pages 5285–5294, 2017.

Learning Map-Independent Evaluation Functions for Real-Time Strategy Games

Zuozhi Yang
Drexel University
Philadelphia, PA, USA
zy337@drexel.edu

Santiago Ontañón
Drexel University
Philadelphia, PA, USA
so367@drexel.edu

***Abstract*—Real-time strategy (RTS) games have drawn great attention in the AI research community, for they offer a challenging and rich testbed for both machine learning and AI techniques. Due to their enormous state spaces and possible map configurations, learning good and generalizable representations for machine learning is crucial to build agents that can perform well in complex RTS games. In this paper we present a convolutional neural network approach to learn an evaluation function that focuses on learning general features that are independent of the map configuration or size. We first train and evaluate the network on a winner prediction task on a dataset collected with a small set of maps with a fixed size. Then we evaluate the network's generalizability to three set of larger maps. by using it as an evaluation function in the context of Monte Carlo Tree Search. Our results show that the presented architecture can successfully capture general and map-independent features applicable to more complex RTS situations.**

***Index Terms*—real-time strategy, neural networks, MCTS**

I. Introduction

Real-time strategy games (RTS) pose a significant challenge to AI research given their huge space space and decision space complexity and their real-time nature. Specifically, an open problem is how to define evaluation functions that, given a game state, predict the probability of each player of winning the game. Previous work in computer Go [1] and RTS games [2] has shown the potential of neural networks to automatically learn such evaluation functions from data. One challenge, however, is how to represent the game state to present it to the network. Existing work [1], [2] has focused on fixed-size games (like Go), or RTS situations with fixed map sizes.

In this paper, we focus on the problem of learning evaluation functions for arbitrary RTS game maps by learning general and map-independent features applicable to complex situations. This is important, since scenario variability is not just a key feature of most RTS games, but of many real-world problems as well.

Specifically, we propose a convolutional neural network architecture specifically designed to learn features that are map size-independent and generalizable to larger maps. We report experiments training the network with data collected on smaller maps, which is cheaper to collect than data for larger maps, and then evaluate the performance of the trained network on a collection of larger maps. By utilizing generalizable features learned from small maps, we can improve the time and data efficiency of our learning approach for building agents for complex maps. All reported experiments are carried out on the μRTS simulator [3].

The remainder of this paper is organized as follows. First, Section II provides some necessary background on RTS AI research, and on μRTS. After that, Section III describes our processes of data collection, neural network architecture, and training procedure. Section IV then reports experimental results under two separate conditions (winner prediction accuracy and in-game playing strength). Finally, the paper closes with conclusions and possible future work.

II. Background

RTS games have drawn great attention in the AI research community, since they offer a challenging and rich testbed for both machine learning and search techniques. For example, many platforms and tools, such as TorchCraft [4], SC2LE [5], μRTS [3], and ELF [6] have been developed to promote research in the area. Specifically, in this paper, we focus on the problem of game state evaluation and use μRTS as our experimental domain, as it offers a minimalist yet sufficient RTS game environment and a collection of implementations of search algorithms to be used for testing. Game state evaluation is an important component of almost any game tree search approach such as minimax or Monte Carlo Tree Search.

Monte Carlo Tree Search (MCTS) [7] algorithms initially became popular because of their success in computer Go, where MCTS was a significant step forward toward overcoming the huge search space of this game via sampling sequences of stochastic play. However, without a good representation of Go knowledge, MCTS failed to scale to the level of top professional Go players. The recent progress that ultimately culminated in defeating professional players [1] has been due to integrating convolutional neural networks (CNN) into the MCTS search process, which can learn complex representations of the game state and perform tasks such as game state evaluation or predicting good moves, in order to guide the search.

In the context of RTS games (and besides non-determinism and partial observability, which are not considered in this paper), two of the main open challenges for game tree search approaches are: (1) the combinatorial branching factor and (2) the lack of accurate state evaluation functions. For example,

978-1-5386-4359-4/18/$31.00 ©2018 IEEE

a game situation where a player controls just 10 units, where each unit can execute 5 different actions each results in a potential branching factor of $5^{10} \approx 10$ million, beyond what standard MCTS algorithms can handle. Additionally, unlike games like Go which are always played from the same starting position, RTS games are played in a large variety of maps that are highly configurable both in terms of size and initial states. Thus, an evaluation function that is not only accurate but also applicable to different maps is key for guiding MCTS.

In this paper, we employ a variant of MCTS algorithms called *NaïveMCTS* [8] to handle the combinatorial branching factor, and focus on the problem of using CNNs to achieve evaluation functions that can generalize to a large variety of maps. There has been successful applications of combining CNN with MCTS in the domain of game AI, such as computer Go [1] and Atari 2600 games [9]. Specifically in the context of RTS games, [2][10] trained a CNN as state evaluation function and combined with Monte Carlo Tree Search, but was limited to fixed sized maps. In this paper we specifically focus designing a CNN approach that generalizes to maps of arbitrary size. Another related piece of work is that of Vinyals *et al.* [5] where, in order to train a map-size independent network, they reduce the resolution of the map and minimap to always be 64x64. In contrast, our approach does not require reducing the input to a fixed size. Finally, Stanley and Miikkulainen [11] proposed an approach based on a "roving eye" for evaluating Go boards of arbitrary sizes. Their approach is based on a fixed size "eye" that moves around the board. They key problem is learning the policy to control the eye to move to the important parts of the board, as well as having a memory that keeps a record of the important aspects it has observed. Our approach does not require learning such policy, and can observe the whole board/map directly (notice that the idea of the roving eye and that of a convolutional neural network are not that different in that both scan smaller neural networks through the input).

A. μRTS

μRTS[1] is a simple RTS game designed for testing AI techniques. μRTS provides the essential features that make RTS games challenging from an AI point of view: simultaneous and durative actions, combinatorial branching factors and real-time decision making. The game can be configured to be partially observably and non-deterministic, but those settings are turned off for all the experiments presented in this paper. We chose μRTS, since in addition to featuring the above properties, it does so in a very minimalistic way, by defining only four unit types and two building types, all of them occupying one tile, and there is only one resource type. Additionally, as required by our experiments, μRTS allows maps of arbitrary sizes and initial configurations.

There is one type of environment unit (minerals) and six types of units controlled by players, which are:

- Base: can train Workers and accumulate resources
- Barracks: can train attack units
- Worker: collects resources and construct buildings
- Light: low power but fast melee unit
- Heavy: high power but slow melee unit
- Ranged: long range attack unit

Additionally, the environment can have walls to block the movement of units. A example screenshot of game is shown in Figure 1. The squared units in green are Minerals with numbers on them indicating the remaining resources. The units with blue outline belong to player 1 (which we will call *max*) and those with red outline belong to player 2 (which we will call *min*). The light grey squared units are Bases with numbers indicating the amount of resources owned by the player, while the darker grey squared units are the Barracks. Movable units have round shapes with grey units being Workers, orange units being Lights, yellow being Heavy units (now shown in the figure) and blue units being Ranged.

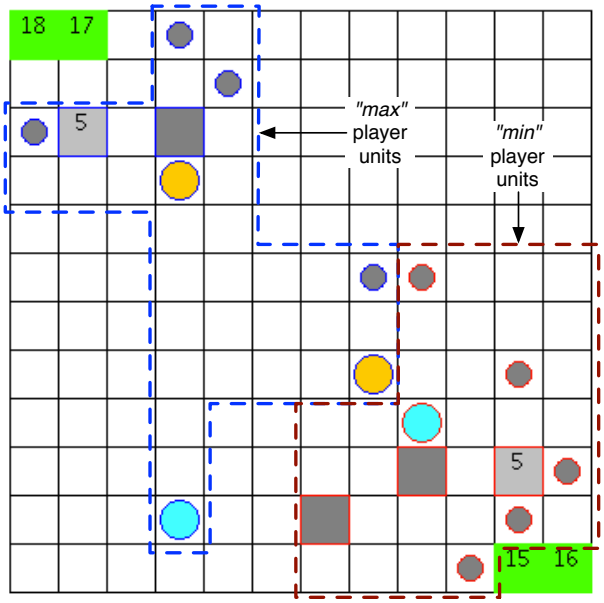


Fig. 1. A Screenshot of μRTS.

III. Learning Transferable Feature using Convolutional Neural Networks

Many deep neural network architectures trained for computer vision tasks learn features similar to Gabor filters and color blobs [12] in the first layer. These features tend to be generalizable to different domains and tasks. Based on this observation, in this paper we design and train our μRTS state evaluation networks for small game maps and show that they generalize to larger maps.

The remainder of this section, first presents the procedure we followed to collect training data for our models, how are the game states encoded to be provided as input to the neural networks, and finally, we describe the proposed network architecture and training procedure.

[1]https://github.com/santiontanon/microrts

A. Data Collection

We collected replay data from a round-robin tournament with 10 μRTS bots. Specifically, we used the following bots:

- *Rush bots:* hardcoded deterministic bots that implement a rush strategy that constantly produces one type of unit and sends them to attack the opponent's base. Specifically, we used the *WorkerRush*, *LightRush*, *RangedRush* and *HeavyRush* bots of μRTS, which implement rushes with worker, light, ranged, and heavy units respectively.
- *PortfolioAI:* playouts using a set of simple bots (WorkerRush, LightRush, RangedRush, and Random), are run all possible pairings, and the minimax bot is selected to produce the next action.
- *MonteCarlo:* flat Monte Carlo search, splitting the computation budget uniformly among all possible actions.
- *ϵ-Greedy MonteCarlo:* same as flat Monte Carlo search but using ϵ-greedy strategy to allocate computation budget to each action.
- *DownsamplingUCT:* Standard UCT [7], but in those nodes of the tree where the branching factor is larger than a constant k ($k = 100$ in our experiments), k of those actions are sampled at random, and only those are considered during search.
- *UCTUnitActions:* Standard UCT, but in nodes where we can issue actions to more than one unit, rather than considering the whole combinatorics, we only consider the actions of one unit per node (the children nodes of this node will consider actions of the rest of units, in turn). This follows the UCT implementation of Balla and Fern in their work on the *Wargus* RTS game [13].
- *NaïveMCTS:* MCTS algorithm with a tree policy specifically designed for games with combinatorial branching factors. The tree policy exploits Combinatorial Multi-Armed Bandits [8] to handle the combinatorial explosion of possible moves.

Four different configurations of NaïveMCTS were used, for a total of 13 different bots. For bots with configurable computing budget (PortfolioAI, MonteCarlo, DownsamplingUCT, UCTUnitActions and NaïveMCTS), we run tournaments under eight different budgets per game frame: $100ms$, $200ms$, $400ms$, $800ms$, 100 playouts, 200 playouts, 400 playouts and 800 playouts (i.e., giving the bots a certain amount of milliseconds per game frame, or a certain number of playouts, which makes sense in the MCTS-based bots).

Maps of size 8×8 with 23 different starting configurations were used. We decided to collect data only on small 8×8 maps, since it's cheaper to generate (games in larger maps take longer to execute). Additionally, in this way, we can also test if the learned evaluation function generalizes to larger maps, when trained only in smaller maps. After we discard all replays that resulted in a draw and duplicated replays from games between two deterministic bots (to avoid biased data), this resulted in 25,200 replays.

Since states coming from the same replay are highly correlated (a state at time t and one at time $t + 1$ from the same replay are likely to be very similar), if we were to use all the states from all replays, the i.i.d. assumption of supervised learning algorithms would be violated. To mitigate this problem, we randomly sample three positions in each replay. We also augmented each position in the training set with all reflections and rotations. As a result, we ultimately have a training set with 484,800 game states and testing set with more than 15,000 game states (if one state extracted from a trace was placed in the training set, then all the other states from that trace were also placed in the training set). Each game state is labeled with the player that won the corresponding game.

TABLE I
DESCRIPTION OF FEATURE PLANES FOR GAME STATE REPRESENTATION

Features	Planes	Description
Unit Type & Position	7	Base, Barracks, Resource, worker, light, ranged, heavy
Unit Health	5	1, 2-3, 4-5,6-7, ≥ 8
Unit Owner	2	Mask each unit to its player owner
Worker Resources	1	1 if the worker is carrying a resource, 0 otherwise

B. Features

For this work, we assumed each map can have an arbitrary size $w \times h$ where w and h is the width and height of the map. Each game state is thus converted into a $15 \times w \times h$ tensor, composed of 15 feature planes (each of them of size $w \times h$), to be fed as input to the neural network. The feature planes that we use (see Table I) come directly from the raw representation of the board information (unit types, unit health, unit owners, unit positions, and resources carried by workers). Many of the features are split into multiple planes of binary values, for example in the case of unit positions there are six feature planes for each type of the units indicating position information. In addition to the 15 feature planes, we have a *global feature array*, consisting of the following features:

- Resources owned by each player
- Proportion of game time left scaled from 0-20

For example a *global feature array* $[5, 6, 4]$ can be interpreted as that the max player has 5 resources, the min player has 6 resources, and the game has about 20% of its maximum pre-defined length (5000 cycles in our experiments) left.

In comparison to the input of Stanescu's work [2], we employ a similar one-hot encoding for spatial information. The difference between our work and theirs is that we separated the global information from the spatial information and they encoded them all together. In next section, we will describe the design of our model and usage of the inputs.

C. Neural Networks Architecture and Training

Since our goal is to train a neural network that can handle different map sizes as input and generalize across them, we made the following design decisions:

1) Structurally: we employed a convolutional neural network (CNN) approach [14]. Usually, CNNs consist of

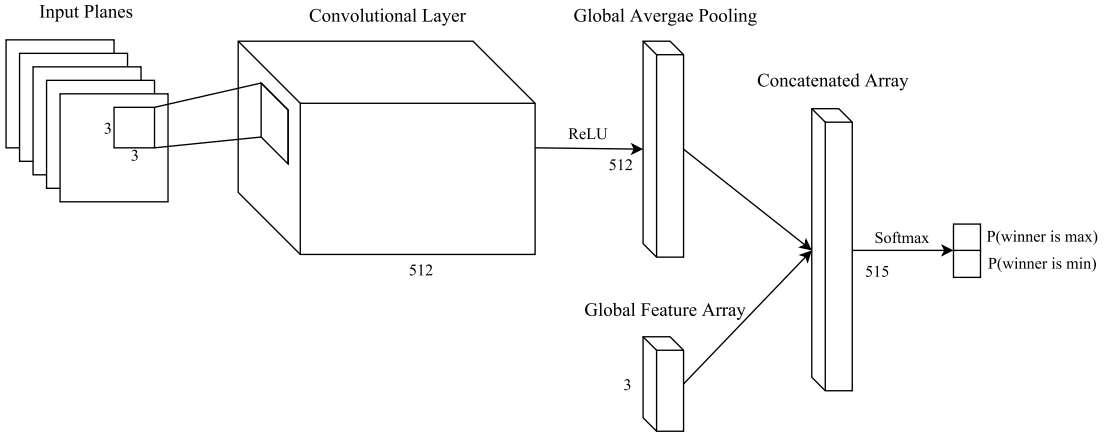


Fig. 2. Proposed neural network architecture for maps-size independent game state evaluation.

a set of *convolutional layers*, followed by a set of fully connected layers. This assumes a fixed input size. We replaced the fully connected layers (typically at the end) by a global average pooling layer (described below), which converts different-sized convolutional feature maps into a fixed size vector. In this way, we can provide maps of different sizes as input.

2) Algorithmically: only using one but thick convolutional layer with 512 filters to focus on learning general features. After this layer, we feed the concatenation of the output of the convolutional layer (after having been converted to a fixed size vector by the pooling layer) and the global feature array to the output layer (as shown on Figure 2).

As described above, the input is the a stack of $15 \times w \times h$ spatial features where $w \times h$ is the size of the input map plus a global feature array of size 3. First, each spatial feature plane is padded with 0s to obtain a $(w+1) \times (h+1)$ plane. Then the input is convolved with 512 3×3 filters with stride 1. The resulting feature maps are also padded with 0s to keep the same size as the input. After the convolutional layer, rectified linear units (ReLU) are applied to calculate activations and a 0.5 dropout ratio is applied to prevent overfitting. We then directly output the spatial average of the feature maps from the convolutional layer as the confidence of categories through a global average pooling layer [15] instead of adopting the fully connected layers for classification that are usual in CNNs. This results on a vector of size 512 (one per spatial feature map). This vector is then concatenated with the non-spatial global feature array. Finally, the concatenated vector is fed into the softmax layer of 2 units, which predicts the winning probability for each player. The final architecture of the proposed neural network is shown in Figure 2.

We use Xavier algorithm to initialize weights [16], and adaptive moment estimation (ADAM) [17] with hyperparameters $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\alpha = 0.001$ as the optimizer. We employ cross entropy loss for backpropagation. The whole training process takes about 10 minutes to converge on a NVIDIA GTX 1080 Ti GPU with the dataset of 484,800 training instances.

IV. Experimental Evaluation

In order to evaluate our approach, we conduct two evaluation experiments:

1) *winner prediction*: we compare the winner prediction accuracy on the collected dataset on 8×8 maps using a test set of 15,000 game states) between the neural network and several human-designed baseline evaluation functions.
2) *game play*: we evaluate the game play strength of a Monte Carlo Tree Search bot using our proposed evaluation function versus using the baseline evaluation functions on three sets of maps of different sizes: 10×10, 12×12, and 16×16.

A. Experimental Setup

In both experiments, we use two simple human-designed evaluation functions and the Lanchester [18] model as the baselines, all of which evaluate the game states by assigning a numerical score to each player and comparing these values. These evaluation functions work as follows:

- *Simple evaluation function*[2]: The scoring system is shown in equation (1). Given a state s and a player p , U_p is the set of units owned by p in state s, R_p is the total amount of resources for player p, $W_p \subseteq U_p$ is the set of player p's workers is s, R_u is the amount of resources each worker unit u is carrying, C_u is the cost of unit u, HP_u the current health points of unit u, and $MaxHP_u$ its maximum health points of u. W_{res}, W_{work}, W_{unit} are constant weights from human experience, which are 20, 10, and 40 respectively (we omit the state s in the equations for simplicity).

$$E_p^S = W_{res} R_p + W_{work} \sum_{u \in W_p} R_u + W_{unit} \sum_{u \in U_p} \frac{C_u HP_u}{MaxHP_u} \quad (1)$$

[2]This corresponds to the *SimpleEvaluationFunction* function of μRTS.

Then the state evaluation for player p ($p \in \{0, 1\}$) is

$$Simple_p = E_p^S - E_{1-p}^S$$

- *SimpleSqrt evaluation function*[3]: The scoring system is very similar to *Simple evaluation function* but takes a square root on the term for unit health.

$$E_p^Q = W_{res} R_p + W_{work} \sum_{u \in W_p} R_u + W_{unit} \sum_{u \in p} \sqrt{\frac{C_u HP_u}{MaxHP_u}} \quad (2)$$

Also, the final evaluation is normalizaed between 0 to 1.

$$SimpleSqrt_p = 2E_p^Q / (E_p^Q + E_{1-p}^Q) - 1$$

- *Lanchester evaluation function*[4]: The Lanchester model is a strong heuristic evaluation model in RTS games since it can be optimized using logistic regression according to specific situations. In Equation 3, α_u is the strength value for the type of unit u, and o is the Lanchester attrition order. We employ the parameters and weights trained by Stanescu *et al.* [2] for our dataset which are optimized for 8×8 maps. Then again, the state evaluation score for player p is calculated as $Lanchester_p = E_p^L - E_{1-p}^L$.

$$\begin{aligned} E_p^L = \; & W_{res} R_p + W_{work} \textstyle\sum_{u \in W_p} R_u + \\ & W_{base} \frac{HP_{base}}{MaxHP_{base}} + W_{barracks} \frac{HP_{barracks}}{MaxHP_{barracks}} + \\ & \textstyle\sum_{u \in U_p} \alpha_u \frac{C_u HP_u}{MaxHP_u} |U_p|^{o-1} \end{aligned} \quad (3)$$

As for the neural network evaluation function, it calculates the state evaluation score for player p by simply taking the difference of winning probability of the output neurons for the both players.

B. Winner Prediction Evaluation

In the first experiment we evaluate our model by comparing the winner prediction accuracy with the baseline evaluation functions. We use the trained neural network and the baseline evaluation functions to predict the final winner of the instances in the test set (see Section III-A). Figure 3 plots the prediction accuracy for all the compared evaluation functions as a function of how close the game was to the end, with 0 being the beginning of the game, and 100 being the end of the game. To calculate this plot, we aggregated the results in 5% buckets.

First, we can see that the performance of baselines are very close to each other, especially at the beginning. Towards the end, Lanchester and Simple seem to edge SimpleSqrt. Our neural network consistently outperforms the baselines, especially at the beginning of the game, which usually has a significant impact on the game, and it is when the prediction task is harder. It is worth noting that we are using the same parameters and weights for Lanchester model from Stanescu *et al.*'s work [2], where they specifically optimized for the dataset used in their experiments. In their work, they report the Lanchester model to significantly outperform the Simple and SimpleSqrt baselines, which we do not observe in our evaluation. This indicates that there might be some amount of overfitting in their training procedure towards the maps used in that work, and that this evaluation function does not seem to generalize well to our dataset which includes a larger variety of maps, and AIs playing with different amounts of computatinoal budget. Also interestingly, Simple seems to outperform SimpleSqrt, even if in the literature of RTS games it has been shown several times [19] that using the square root of the hitpoints of units results in stronger evaluation functions (as SimpleSqrt does). This does not seem to be the case in μRTS.

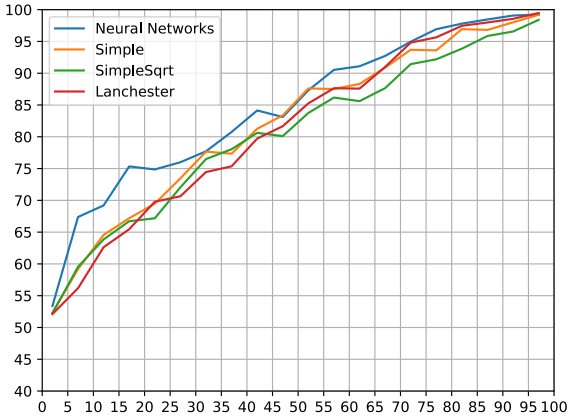


Fig. 3. Winner prediction accuracy comparison. Vertical axis shows prediction accuracy, and horizontal axis shows game time (as a percentage of total length of a replay).

C. Gameplay Strength Evaluation

The second, and perhaps more important, experiment is to test the neural network evaluation function in actual game play under maps of different sizes. In this experiment, we have 10 maps with different initial states for each testing size (10×10, 12×12, and 16×16). Examples of maps of each size is given in Figure 4. Thus in total we have 30 maps for testing. We run a round-robin tournament between the neural network evaluation function and three baseline evaluation functions, all of which are coupled with *NaïveMCTS* with the same parameter configuration (the default parameters in μRTS for this algorithm). Each pair of bots play four games against each other in the tournament per map and are given the same computational budgets for both. Therefore, every pair of bots will play $10 \times 3 \times 4 = 120$ games.

We performed these experiments limiting the computational budget in two different ways: (1) the first is giving each bot only 200 playouts per game frame, and (2) giving each bot only 200 milliseconds per game frame. We use the win rate as the metric for game play strength. If there is a draw, both bots will receive a 0.5 win. The winning rate results are reported in figures 5 and 6 and the win/draw/loss matrix for each bot pair is reported in tables II and III.

As we can observe from the results, when limiting the computation budget by the number of playouts, the neural network

[3]This corresponds to the *SimpleSqrtEvaluationFunction3* function of μRTS.
[4]This corresponds to the *LanchesterEvaluationFunction* function of μRTS.

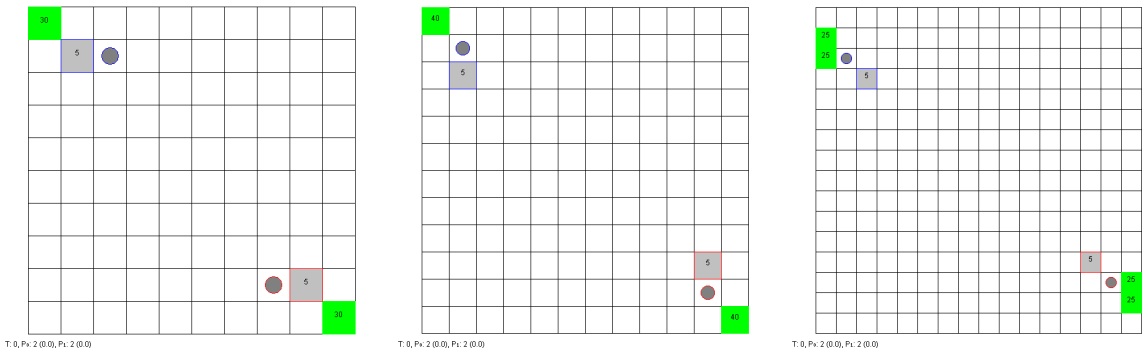


Fig. 4. Example 10×10, 12×12 and 16×16 maps used in our experimental evaluation.

has a significant advantage over the baselines on gameplay strength. We can also see that SimpleSqrt and Lanchester perform particularly poorly when compared to the neural network. Additionally, it might seem that the performance of the neural network degrades when in 16x16 maps as compared to 12x12 maps. However, notice that what happens is that in 16x16 maps there are a lot of ties, since the map is large and some times players do not find each other. However, in 16x16 maps, as Table 5 shows, the neural network only lost 4 (out of 40) games against Simple, 1 against SimpleSqrt and 0 against Lanchester. This is a limitation of using playouts of length 100 in *NaïveMCTS*, rather than of the evaluation functions.

Concerning limiting the computation budget by time (as would happen in real world settings), the Simple evaluation outperformed the neural network. This is due to two main reasons. First, the intrinsic reason that the neural network works less dominantly is that the time needed for neural network to evaluate a single state is longer than that required by the baseline evaluation functions. However, a second significant contributing factor is that μRTS is written in Java and the neural network model is written and trained in Python. Thus on average the evaluation function spent around 37% of time budget on communication between the languages, which happens via a socket. We believe, that if this communication time is removed, the problem can be mitigated and our model will outperform all the baselines. In our experiments, the average ratio of playouts completed with in the same time budget between the neural network and baselines are around 16:1, 13:1, and 9:1, respectively for maps of size 10×10, 12×12, and 16×16. If the communication time is eliminated, these ratios would go down to about 10:1, 8:1 and 5.5:1. Possible solutions to address this problem are discussed below.

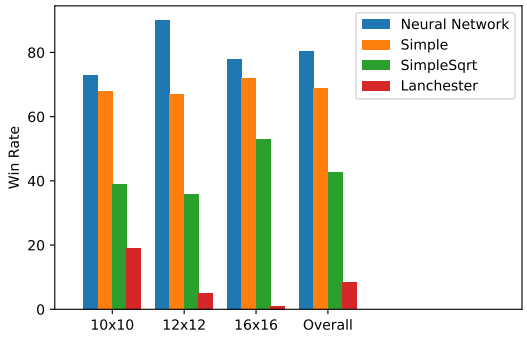


Fig. 5. Gameplay strength results with playout budget

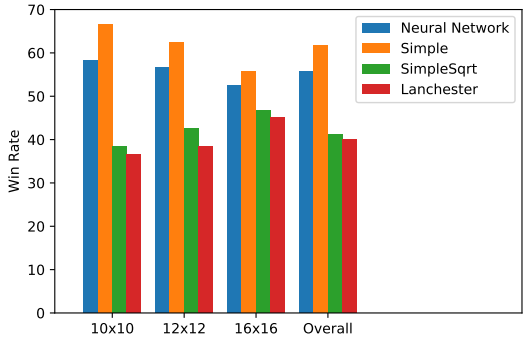


Fig. 6. Gameplay strength results with time budget

V. Conclusion and Future Work

This paper presents a convolutional neural network architecture designed for the task of game state evaluation in real-time strategy games, and specifically to address the problem of training a network that can be used (without retraining) in arbitrarily sized maps. Our proposed neural network state evaluation function can be trained on size-independent data and can perform well on maps that are more complex than those used for training (notice that although we tested the network in maps up to 16×16, all training data came from 8×8 maps). Structurally, replacing the traditionally used fully connected layers by a global average pooling layer allows the neural network to take varied-sized inputs. Moreover, we only employ one convolutional layer to learn generalizable features and prevent overfitting on the training map size.

Our empirical results showed that we have successfully learned features from game replays collected under simple

TABLE II
GAMEPLAY STRENGTH RESULTS (WIN/DRAW/LOSS OF THE ROW AGENT) WITH PLAYOUT BUDGET.

Result Matrix for 10 × 10 Maps				
	Neural Network	Simple	SimpSqrt	Lanchester
Neural Network	N/A	19/12/9	29/5/6	22/12/6
Simple	9/12/19	N/A	28/3/9	26/11/3
SimpSqrt	6/5/29	9/3/28	N/A	17/6/17
Lanchester	6/12/22	3/11/26	17/6/17	N/A

Result Matrix for 12 × 12 Maps				
	Neural Network	Simple	SimpSqrt	Lanchester
Neural Network	N/A	21/9/10	35/2/3	35/3/2
Simple	10/9/21	N/A	28/4/8	36/2/2
SimpSqrt	3/2/35	8/4/28	N/A	30/3/7
Lanchester	2/3/35	2/2/36	7/3/30	N/A

Result Matrix for 16 × 16 Maps				
	Neural Network	Simple	SimpSqrt	Lanchester
Neural Network	N/A	13/23/4	12/27/1	40/0/0
Simple	4/23/13	N/A	22/14/4	40/0/0
SimpSqrt	1/27/12	4/14/22	N/A	38/2/0
Lanchester	0/0/40	0/0/40	0/2/38	N/A

TABLE III
GAMEPLAY STRENGTH (WIN/DRAW/LOSS OF THE ROW AGENT) RESULTS WITH TIME BUDGET

Result Matrix for 10 × 10 Maps				
	Neural Network	Simple	SimpSqrt	Lanchester
Neural Network	N/A	14/4/18	22/6/12	20/14/6
Simple	18/4/14	N/A	28/6/6	22/14/4
SimpSqrt	12/6/22	6/6/28	N/A	16/8/16
Lanchester	6/14/20	4/14/22	16/8/16	N/A

Result Matrix for 12 × 12 Maps				
	Neural Network	Simple	SimpSqrt	Lanchester
Neural Network	N/A	12/12/16	22/8/10	20/8/12
Simple	16/12/12	N/A	22/6/12	24/8/8
SimpSqrt	10/8/22	12/6/22	N/A	20/4/16
Lanchester	10/8/22	24/8/8	16/4/20	N/A

Result Matrix for 16 × 16 Maps				
	Neural Network	Simple	SimpSqrt	Lanchester
Neural Network	N/A	10/18/12	16/14/10	12/18/10
Simple	12/18/10	N/A	10/20/10	16/20/4
SimpSqrt	10/14/16	10/20/10	N/A	6/26/8
Lanchester	10/18/12	4/20/16	8/26/6	N/A

maps that can be transferred to larger maps. Specifically, the result of gameplay strength under time budget showed promise, but also showed that there are still challenges to be addressed. The main challenge is that the evaluation of the neural network is slower than the baseline evaluation functions used in our experiments. Possible solutions involve using smaller networks (we are currently using 512 filters, which could be reduced trading accuracy by speed). Another idea to explore further is to better exploit the parallel power of the GPU, for example, exploiting existing work on parallelizations of the MCTS search algorithm [20], the evaluation function could be executed in batches, rather than state by state, shortening the speed gap between neural networks model and human-engineered evaluation functions. Finally, we would like to study how to extend our ideas to more realistic partially observable settings.

REFERENCES

[1] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[2] M. Stanescu, N. A. Barriga, A. Hess, and M. Buro, "Evaluating real-time strategy game states using convolutional neural networks," in *Computational Intelligence and Games (CIG), 2016 IEEE Conference on*. IEEE, 2016, pp. 1–7.

[3] S. Ontañón, "The combinatorial multi-armed bandit problem and its application to real-time strategy games," in *Ninth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2013.

[4] G. Synnaeve, N. Nardelli, A. Auvolat, S. Chintala, T. Lacroix, Z. Lin, F. Richoux, and N. Usunier, "Torchcraft: a library for machine learning research on real-time strategy games," *arXiv preprint arXiv:1611.00625*, 2016.

[5] O. Vinyals, T. Ewalds, S. Bartunov, P. Georgiev, A. S. Vezhnevets, M. Yeo, A. Makhzani, H. Küttler, J. Agapiou, J. Schrittwieser *et al.*, "StarCraft II: a new challenge for reinforcement learning," *arXiv preprint arXiv:1708.04782*, 2017.

[6] Y. Tian, Q. Gong, W. Shang, Y. Wu, and C. L. Zitnick, "Elf: An extensive, lightweight and flexible research platform for real-time strategy games," *Advances in Neural Information Processing Systems (NIPS)*, 2017.

[7] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of monte carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in games*, vol. 4, no. 1, pp. 1–43, 2012.

[8] S. Ontanón, "Combinatorial multi-armed bandits for real-time strategy games," *Journal of Artificial Intelligence Research*, vol. 58, pp. 665–702, 2017.

[9] X. Guo, S. P. Singh, R. L. Lewis, and H. Lee, "Deep learning for reward design to improve monte carlo tree search in atari games," in *IJCAI*, 2016.

[10] N. A. Barriga, M. Stanescu, and M. Buro, "Combining strategic learning with tactical search in real-time strategy games," in *Proceedings of the Thirteenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE-17), October 5-9, 2017, Snowbird, Little Cottonwood Canyon, Utah, USA.*, 2017, pp. 9–15. [Online]. Available: https://aaai.org/ocs/index.php/AIIDE/AIIDE17/paper/view/15814

[11] K. O. Stanley and R. Miikkulainen, "Evolving a roving eye for go," in *Genetic and Evolutionary Computation Conference*. Springer, 2004, pp. 1226–1238.

[12] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?" in *Advances in neural information processing systems*, 2014, pp. 3320–3328.

[13] R.-K. Balla and A. Fern, "Uct for tactical assault planning in real-time strategy games." in *IJCAI*, 2009, pp. 40–45.

[14] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[15] M. Lin, Q. Chen, and S. Yan, "Network in network," *arXiv preprint arXiv:1312.4400*, 2013.

[16] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 2010, pp. 249–256.

[17] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[18] M. Stanescu, N. Barriga, and M. Buro, "Using lanchester attrition laws for combat prediction in starcraft," in *Eleventh Annual AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 2015, pp. 86–92.

[19] A. Kovarsky and M. Buro, "Heuristic search applied to abstract combat games," in *Conference of the Canadian Society for Computational Studies of Intelligence*. Springer, 2005, pp. 66–78.

[20] G. M.-B. Chaslot, M. H. Winands, and H. J. van Den Herik, "Parallel monte-carlo tree search," in *International Conference on Computers and Games*. Springer, 2008, pp. 60–71.

Imitation Learning with Concurrent Actions in 3D Games

Jack Harmer[1], Linus Gisslén*[1], Jorge del Val*[1], Henrik Holst[1], Joakim Bergdahl[1], Tom Olsson[2], Kristoffer Sjöö[1], Magnus Nordin[1],

***Abstract*—In this work we describe a novel deep reinforcement learning architecture that allows multiple actions to be selected at every time-step in an efficient manner. Multi-action policies allow complex behaviours to be learnt that would otherwise be hard to achieve when using single action selection techniques. We use both imitation learning and temporal difference (TD) reinforcement learning (RL) to provide a 4x improvement in training time and 2.5x improvement in performance over single action selection TD RL. We demonstrate the capabilities of this network using a complex in-house 3D game. Mimicking the behavior of the expert teacher significantly improves world state exploration and allows the agents vision system to be trained more rapidly than TD RL alone. This initial training technique kick-starts TD learning and the agent quickly learns to surpass the capabilities of the expert.**

I. Introduction

In recent years the field of reinforcement learning (RL) [Sutton and Barto, 1998] has undergone a renascence, due to the transformative powers of deep neural network architectures. A key strength of these architectures is their ability to be used as arbitrary function approximators. This ability has allowed neural network based model free RL techniques to solve a number of challenging tasks that were previously intractable. The techniques described by Mnih et al. [2015] demonstrate the power of this, in their work they develop an algorithm (DQN) that employs a neural network to estimate the value of high dimensional input states and use a bootstrapping technique to train it. Put simply, this algorithm minimizes the difference between the networks estimate of the value of the current state and that of a target value, where the target value is simply the networks predicted value of the next state plus any rewards that were received in-between the two states. The target value is more grounded in reality than the initial guess because it is partly made up of rewards gained through an agents interactions with the environment, and as such they show that these updates allow the network to learn an accurate value function. They further demonstrate the capability of this algorithm by training agents to play Atari 2600 video games using high dimensional raw pixel values as the input state. Due to the success of this work and a renewed interest in the field, a large number of improvements to this algorithm have now been suggested in the literature, including but not limited to: A modification that reduces the bias of the value function estimate (DDQN) [van Hasselt et al., 2015], a technique for improving the data efficiency of the algorithm, by adding a type of prioritisation to the experience replay memory sampling scheme [Schaul et al., 2015], adding noise to specific layers to improve exploration [Fortunato et al., 2017], breaking the action value function into two components, one that models the value of the state and one that models the per-action advantage [Wang et al., 2016], and also modeling the state value function as a distribution [Bellemare et al., 2017].

Unlike these previous off-policy techniques, Mnih et al. [2016] propose an algorithm that moves away from an experience replay based training regime. Instead, they describe an architecture (A3C) that performs updates using data from a large number of simultaneously running agents. They show that training using multiple agents, each with their own version of the environment, decorrelates updates in a similar manner to memory sampling in DQN, with the added benefit of improved exploration, training speed and stability.

Despite these advances, algorithms based around temporal difference (TD) RL are computationally expensive and can take a significant amount of time to train. Training using TD RL is only effective if the target value is more grounded in reality than the current estimate. This condition is only satisfied when there is a net accumulation of reward between states, otherwise training simply updates one guess towards another. Thus training using TD RL is particularly problematic in reward sparse environments where it might require many specific consecutive actions to receive a reward. Consider a racing car game where the reward is scaled inversely with lap time and received after completing a lap. This task is extremely difficult to solve when using TD RL. To receive a reward and thus perform one useful update, an agent would have to select the correct action for many thousands of steps at a stage when the agent has no understanding of the world.

The technique known as reward shaping [Ng et al., 1999] can alleviate some of the problems with reward sparse environments. When using this technique the reward function of the task is changed by an expert, who understands the objective, in order to encourage behaviours that help the agent solve the task. However, great care has to be taken because it is not always trivial to tweak the rewards without significantly altering the nature of the task at hand. It is also often difficult to break down a complicated task into a number of smaller sub-tasks amenable to shaping.

Tasks where large action spaces are required are also difficult to train when using TD RL, because the probability

Correspondance to jharmer@ea.com
[1]Electronic Arts, SEED, Stockholm, Sweden
[2]Electronic Arts, DICE, Stockholm, Sweden
*These authors contributed equally

978-1-5386-4359-4/18/$31.00 ©2018 IEEE

of selecting the correct action, in order to receive a reward, decreases as the size of the action space increases. Further, credit assignment also becomes more problematic [Sutton and Barto, 1998].

When training agents to interact in complex environments with large action spaces, the behaviour associated with having a single action per time step (SAPS) policy, as is almost always the case in RL, is often undesirable. For example, running forward whilst strafing and shooting in a video game is an effective strategy that is impossible to achieve when using SAPS architectures. When solving problems where multiple actions per time step are required, most networks architectures rely on modeling all possible combinations of actions as separate distinct actions [Mnih et al., 2015, Wang and Yu, 2016]. However when using large actions spaces, the dramatic increase in the number of possible action combinations severely limits the applicability of such techniques. For example, a typical modern video game controller might have around 20 controls, modeling all possible combinations of these inputs would require a policy which outputs $\sim 10^6$ (2^{20}) probabilities. Joint action representations in such a large action space make it much harder for the agent to learn the value of each of the true actions, and do not take advantage of the underlying relationships between different sets of individual actions. For these reasons, an algorithm that allows multiple output actions per time step (MAPS) should improve the performance of such an agent.

A powerful technique that can be used to speed up training is to teach by example. The idea being that instead of using a domain expert to break down rewards into more fine-grained rewards, an expert can be used to demonstrate the desired behaviour. Then, the network is left to determine how to change its policy in order to match the expert behaviour. This technique is known as imitation learning (IL) [Subramanian et al., 2016, Hester et al., 2017a, Le et al., 2018, Andersen et al., 2018, Nair et al., 2017, Zhang and Ma, 2018, Gao et al., 2018]. Imitation learning provides the agent with prior knowledge about effective strategies for behaving in the world. Combining TD RL with IL allows an agent to learn from it's own experiences, and helps to avoid situations where the skill of an agent is limited by the skill of the teacher. Learning via imitation can either be the goal itself, or an auxiliary task that is used to help achieve another goal by bootstrapping off the behaviour of an expert

II. Contributions

Motivated by the goal of adding neural network controlled AI agents to future games, in order to increase levels of player immersion and entertainment, we describe a technique for training an agent to play a 3D FPS style game. In comparison to 2D games such as those on the Atari 2600 platform, 3D FPS games are a particularly challenging problem for RL. This is mainly due to the factors described previously but also because of the partially observed nature of such games, and the challenges related to exploring large state spaces. We develop an in house FPS game using a modern game engine in order to test the performance of agents in scenarios with high visual fidelity graphics.

We present an A3C derivative algorithm that combines supervised imitation learning (learning via guidance from an expert teacher) with temporal difference RL (learning via trial and error), throughout training; using only a small amount of expert data. Imitation learning is used as an auxiliary task in order to help the agent achieve it's primary goal, playing the game.

We describe a neural network architecture that outputs multiple discrete actions per time step without having to model combinations of actions as separate actions, and describe a loss function that allows the policy to be trained. Combining multi-action per time step RL with imitation learning in this manner allows higher quality expert data to be used, as it circumvents the difficulties associated with recording expert data when the expert is limited to single action per time step interactions with the environment. We call the resulting model Multi-Action per time step Imitation Learning (MAIL).

- We present a neural network architecture that outputs multiple discrete actions per time step (MAPS), as well as a loss function for training the multi-action policy.
- We describe a technique for training this algorithm using a combination of imitation learning and temporal difference reinforcement learning (MAIL).
- We describe how these techniques can be used to teach an agent to play a challenging fully 3D first person shooter (FPS) style video game, an important milestone on the way to training neural networks to play modern AAA FPS games.

III. Related Work

Training using IL is by definition off-policy and as such is typically limited to off-policy training techniques [Hester et al., 2017b], which tend to be less stable than on-policy algorithms [Mnih et al., 2016], or to being carried out as a pre-training step [Silver et al., 2016, Schaal, 1997]. Indeed in one of the earliest examples of this Schaal [1997] use imitation learning as a pretraining step during which they train the value function and policy, after which they continue to train the network using reinforcement learning alone. However, they found that even for simple nonlinear tasks task-level imitation based on direct-policy/value learning, augmented with subsequent self-learning, did not provide significant improvements to learning speed over pure trial-and-error learning without demonstration. When imitation learning is used as a pre-training step, the policy is also often prone to collapse due to the limited state-space coverage of the expert data; that is, models tend to over-fit to the data instead of learning a general solution. This can be mitigated by using a large amount of expert training data, as in the work by Silver et al. [2016]. However, the time, effort and cost associated with collecting such data is often a limiting factor in the effective deployment of these techniques. Ross et al. [2010] develop a technique called DAGGER that iteratively generates new policies based on polling the expert policy. However, DAGGER requires additional feedback during training and therefor requires the expert to be available, as such it is impractical for long training runs or when access to the expert is limited. Another example of work in

this category, i.e. combining expert data and reinforcement learning, is Approximate Policy Iteration with Demonstration (APID) [Kim and Pineau, 2013]. Here an expert's trajectories are used to define linear constraints which are used in the optimization made by a Policy Iteration algorithm.

Wang and Yu [2016] look at the problem of performing multiple actions per time step in a Q-learning setting by modeling combinations of actions as separate actions and estimating the value for each of these. However as previously mentioned this technique is problematic when using large actions spaces, due to the rapid increase in the number of possible action combinations that have to be accounted for. Moreover, their results are only contextualized in Least Squares Policy Iteration (LSPI) and TD learning with linear value function approximation, rather than an actor critic deep reinforcement learning approach. Sharma et al. [2017] attempt to reduce the combinatorial explosion by reducing the action space into a number of sub-actions that are mutually exclusive for a given scenario, such as the actions representing left and right in a video game. However, this requires specific prior knowledge of the action space and only partly offsets this effect. Furthermore, they still have to evaluate each possible combination separately to select a joint action, which is not efficient for large action spaces. Lillicrap et al. [2015] describe a technique for selecting multiple actions per time step in a continuous action setting, however such techniques are notoriously difficult to train due to their brittleness and hyperparameter sensitivity [Haarnoja et al., 2018].

IV. Preliminaries

We consider a Markov Decision Process (MDP) described by the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, r \rangle$, where $\mathcal{S}$ is a finite set of states, $\mathcal{A}$ is a finite set of actions containing N possible actions, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the transition probability kernel and $r : \mathcal{S} \to \mathbb{R}$ is the reward function. In the standard SAPS context the action space is:

$$\mathcal{A}_{sa} = \{A_1, A_2, ..., A_N\}. \tag{1}$$

Consequently, the action $a_t \in \mathcal{A}_{sa}$ at each timestep t is limited to be one of the available actions. Each action is selected using a stochastic policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$, so that $\mathbf{a}_t \sim \pi(a_t|s_t)$. When using high-dimensional state spaces, these policies are typically parametrized by a deep neural network of weights θ; that is, $p(a_t|s_t) = \pi_\theta(a_t|s_t)$.

In a MAPS setting, we allow the agent to select multiple actions at each timestep. Hence, the action space becomes the space of all possible subsets of different elements of $\mathcal{A}_{sa}$. We can easily see we can represent those combinations with binary vectors of dimension N, i.e., elements of $\mathbb{Z}_2^N$, where each component a_i indicates whether the action $A_i \in \mathcal{A}_{sa}$ was taken or not:

$$\mathcal{A}_{ma} = \{(a_1, a_2, ..., a_N) | a_i \in \{0, 1\}\}. \tag{2}$$

Here, we have added the zero action to indicate that none of the possible actions was taken. Clearly under this framework $\mathcal{A}_{sa} \subset \mathcal{A}_{ma}$, since the SAPS space could just be defined as $\mathcal{A}_{sa} = \{a \in \mathcal{A}_{ma} | \sum_{i=1}^{N} a_i = 1\}$. We denote the marginal probability of a single component a_i with $p(a_i)$.

We let $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$ denote the total cumulative discounted reward. We also let $V^\pi(s) = \mathbb{E}[R_t|s_t = s]$ denote the state value function, $Q^\pi(s, a) = \mathbb{E}[R_t|s_t = s, a_t = a]$ denote the action value function, and $A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s)$ denote the advantage function corresponding to policy π.

In value-based deep reinforcement learning methods, the action value function is approximated by a deep neural network of parameters ϕ, this is, $Q^\pi(s, a) \approx Q^\pi_\phi(s, a)$. In some cases it is a direct estimate of the value function that is approximated $V^\pi(s) \approx V^\pi_\phi(s)$. The parameters of the policy are updated to minimize the loss over each batch of experiences to approximately satisfy Bellman's equation:

$$L_v(\phi) = \frac{1}{K} \sum_{k=1}^{K} \| r_k + \gamma V^\pi_{\phi'}(s'_k) - V^\pi_\phi(s_k) \|^2, \tag{3}$$

where ϕ' denotes the parameters of a separate *target network*, which are clamped in the loss function, and k indexes each experience (s_k, a_k, r_k, s'_k) on a batch containing K experiences. In policy-based methods, by contrast, it is the policy network π_θ that is updated following estimates of the policy gradient

$$\nabla_\theta \mathbb{E}[R_t] = -\mathbb{E}\left[\sum_\tau \nabla_\theta \log \pi_\theta(a_\tau|s_\tau) \Psi^\pi_\tau \right], \tag{4}$$

where the expectation is taken accross the set of all possible paths and Ψ^π_τ can be a variety of choices, among which the advantage function A^π_τ is one of the most common, since it yields almost the lowest possible variance of the gradient estimator [Schulman et al., 2015]. In actor-critic frameworks, the advantage function is estimated using a value network as in value-based methods; this is, $A_t = \hat{R}_t - V^\pi_\phi(s_t)$, where $\hat{R}_t$ is a TD estimate of R_t. Hence, the estimate of the policy gradient over a set of M independent rollouts $B_1, ..., B_M$ is:

$$\nabla_\theta \mathbb{E}[R_t] \approx -\frac{1}{M} \sum_{i=1}^{M} \left(\sum_{t \in B_i} \nabla_\theta \log \pi_\theta(a_t|s_t) A_t \right). \tag{5}$$

V. Algorithm

In a discrete multi-action setting it can easily be seen that the cardinality of the space grows exponentially with the number of available actions as $O\left(2^N\right)$. As a consequence, a policy which models the the probability of all possible set of event becomes intractable as N becomes large. In order to circumvent this problem, in this paper we make the following structural assumption for the policy:

Assumption 1. *Under the policy π, each component a_i of $a \in \mathcal{A}_{ma}$ is conditionally independent given the state s. That is*

$$p(a|s) = \prod_{i=1}^{N} p(a_i|s). \tag{6}$$

This assumption simplifies the problem to the one of modeling N parameters instead of 2^N, making it tractable, at expense of losing representation power and all conditional dependencies. In theory this can be restrictive, since a high

probability of a joint action would necessarily imply high marginal probabilities of single actions, and vice versa. However, we find that in practice the flexibility of this family of policies is enough to outperform single-action policies. The relaxation of this assumption and the exploration of the tradeoff between flexibility and performance is left for future work.

We hence model our policy as a set of Bernoulli random variables whose probabilities $p(a_i) = \phi_{i;\theta}(s_t) \in [0, 1]$ are outputs of a deep neural network. This is

$$\pi_\theta(a_t|s_t) = \prod_{i=1}^{N} a_{i;t}\phi_{i;\theta}(s_t) + (1 - a_{i;t})(1 - \phi_{i;\theta}(s_t)), \quad (7)$$

where $a_{i;t}$ denotes the i_{th} component of a_t. To sample from this distribution we just have to sample each action independently, relying on assumption 1.

As discussed in section IV, an estimate of the policy gradient is

$$\hat{g} = -\frac{1}{M}\sum_{i=1}^{M}\left(\sum_{t\in B_i}\nabla_\theta \log \pi_\theta(a_t|s_t)A_t\right), \quad (8)$$

where A_t is an estimate of the advantage function at time t. In our case, we can easily particularize this expression following equation (7):

$$\begin{aligned}
&\log \pi_\theta(a_t|s_t)\\
&= \log\left(\prod_{i=1}^{N} a_{i;t}\phi_{i;\theta}(s_t) + (1 - a_{i;t})(1 - \phi_{i;\theta}(s_t))\right)\\
&= \sum_{i=1}^{N} \log\left(a_{i;t}\phi_{i;\theta}(s_t) + (1 - a_{i;t})(1 - \phi_{i;\theta}(s_t))\right) \quad (9)\\
&= \sum_{i=1}^{N} a_{i;t}\log(\phi_{i;\theta}(s_t)) + (1 - a_{i;t})\log(1 - \phi_{i;\theta}(s_t))\\
&= -H(a_t, \phi_\theta(s_t)),
\end{aligned}$$

where for convenience we use H to denote the standard cross-entropy formula; although we remark that it is not cross-entropy in an information-theoretic framework, since a_t is not sampled from a categorical distribution. Our proposed gradient update for rollouts $B_1, ..., B_M$ is thus

$$\hat{g} = \frac{1}{M}\sum_{i=1}^{M}\left(\sum_{t\in B_i}\nabla_\theta H(a_t, \phi_\theta(s_t))A_t\right). \quad (10)$$

A. Imitation Learning

TD RL can be highly inefficient when training agents to perform tasks in complex environments with sparse reward and/or high dimensional action spaces. A powerful yet simple technique for improving pure TD learning is to train the network to imitate the behaviour of an expert in the domain, be it another algorithm or a human expert. Silver et al. [2016] describe an effective technique for this and manage to train neural network controlled agents to play the game of Go to superhuman performance levels. They perform imitation learning as a pre-training step before RL. They sample from a large repository of expert human data (30 million examples) and use the data to train a deep neural network to maximise the likelihood of selecting the expert action, given the same input.

One of the major problems associated with pre-training with imitation learning, is over-fitting to the expert data. The network remembers exactly what actions to perform for a specific input image in the expert training data set, instead of learning a robust and general solution to the problem. Then, when new states are encountered during TD learning, the agent is incapable of selecting an action intelligently. Silver et al. [2016] work around this by training using a very large expert data set and are helped by the fully observed nature of the task.

Due to the difficulties involved in collecting a large amount of expert data, we take a different approach. Instead of applying imitation learning as a pre-training step, we apply it at the same time as TD RL as a way of regularizing the TD learning. Each batch update is comprised of both expert and live agent data. At every update step, the network predicts the action of the expert, from a sample of the expert data, whilst learning a policy that maximises the discounted future reward of the live agent stream. Training the network in this way allows the network to maintain a valid TD learning compatible state, throughout training.

To encourage generalisation, we add Gaussian noise to the expert data inputs and apply dropout after every layer, except the outputs. Dropout is only used for the expert data. To prevent the final performance of the agent from being limited by the quality of the expert data, the IL loss weighting factor, λ_E, is linearly decayed from the start of training.

We found that training the value function using the expert data reduced the performance and stability of the agent. As such the expert data was only used to train the policy whereas the value function was trained using pure TD RL alone.

Our final policy update for the MAIL network for a set of M independent rollouts of live experiences $B_1, ..., B_M$ and M independent batches of expert data $B_1^E, ..., B_M^E$ is thus:

$$\hat{g} = \frac{1}{M}\sum_{i=1}^{M}\nabla_\theta\left(\sum_{t\in B_i} H(a_t, \phi_\theta(s_t))A_t + \lambda_E \sum_{e\in B_i^E} H\left(a_e, \phi_\theta(s_e)\right)\right).$$

VI. Experimental Methods

All agents were trained using a batched version of the A3C algorithm (A2C), similar in design to Babaeizadeh et al. [2016], with the addition of the modifications described previously.

The performance of the following algorithms were evaluated:

- SAPS with TD learning
- MAPS with TD learning
- MAIL without TD learning
- MAIL with λ_E decay over 15M steps

Fig. 1: The environment. A: Top down overview of the play area. B: Enemy. C: Region-of-interest (light blue region) and agent. D: Health. E: Ammo

- MAIL with λ_E decay over 50M steps

In the MAIL experiments, each training batch consisted of approximately 50% on-policy live agent data and 50% expert data. Salient information for the expert data is provided in Table I. Expert data was generated prior to training by recording episodes of human play. At each time-step, the following information was stored in a memory buffer: input observation, expert action vector, reward, terminal state and game features vector. The game features vector contained the agents health and ammo to simulate the on-screen text that a human player can read.

TABLE I: Expert data statistics

Observations	$\sim$ 40000
Episodes	30
Mean score	47
Score std	33

An in-house developed 3D FPS video game was used as the training environment. In the game, rewards are received for eliminating enemies, collecting health and ammo and for finding and occupying a region-of-interest on the map. The location of the health, ammo boxes and region-of-interest change at regular intervals throughout each episode to a random location. Enemies spawn in waves and navigate towards the agent, attacking once within range. Figure 1 provides a visual overview of the environment and demonstrates the key features of the game.

At each time step, the agent observes a 128x128 pixel RGB image (see Figure 2) of the agents first-person view. A small short range radar is visible in the bottom left corner of the agents input image. The agent is also provided with a game features vector that contains information related to the agents health and ammo. Experiments indicated that using 128x128 RGB image observations improved the agents performance relative to 84x84 observations, due to the high visual fidelity of the environment.

The range of actions that the agent can perform include 13 distinct actions that control: translation (x, y, z), head tilt, rotation (multiple torque settings), firing, no-op (SAPS tests). In the MAPS experiments, any combination of the actions in the action set can be selected at every step.

TABLE II: Network Architecture

Layer	N	Details
Conv. 1	32	5x5 kernel, stride 2
Conv. 2	32	3x3 kernel, stride 2
Conv. 3	64	3x3 kernel, stride 2
Conv. 4	64	3x3 kernel, stride 1
Linear	256 + 2	2 input features
LSTM	256	
Policy	13	
Value	1	

We used the base network architecture that is shown in Table II for all experiments. The high level input features (ammo and health) were concatenated to the output of the linear layer, prior to the LSTM (see Figure 3). The inputs were normalised by their maximum possible value. Training parameters that were global to all experiments are shown in Table III.

For the IL experiments, Gaussian noise was added to both the input observations (mean 0, std 0.1) and high level features vector (mean 0, std 0.3) Dropout was applied to all hidden and convolutional layers. We used dropout values of 60% and 50% for the convolution and hidden layers respectively. Dropout was not applied when processing live agent data. Dropout was chosen over L_2 weight regularisation, to reduce the risk of the network finding non-optimal local minima [Goodfellow et al., 2016], instead of more general solutions with larger weights. For the experiments using IL decay, the expert prediction loss factor, λ_E, was linearly decayed from 1.0 to 0.0 over the number of decay steps for the experiment.

The main results are shown in Figure 4. SAPS A3C (red curve) reaches a final score of $\sim$ 40. MAPS A3C (blue curve),

TABLE III: Global parameters

Image size	[128,128,3]
Input features size	2
Batch-size	80
Roll-out length	20
Gradient norm clipping	0.5
Optimiser	Adam
Initial learning rate	1e-4
Final learning rate	1e-5
training steps	75e6
λ_E	1

reaches a final score of ~ 25. MAIL (green curve) reaches a final score of ~ 100.

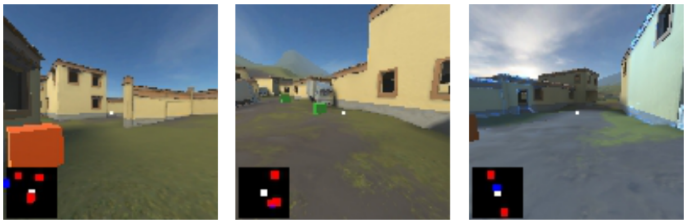


Fig. 2: Example input observations. Left: The agent can see a red health box and some buildings in the main view. The agent can also see a number of red enemies and the blue region-of-interest marker in the radar view. Centre: An example of a green ammo box. Right: The agent has reached the region of interest, indicated by blue lighting on the floor around the agent.

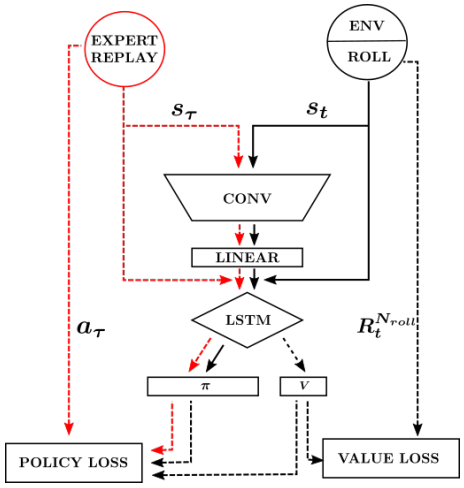


Fig. 3: MAIL Neural network architecture. Solid black lines represent the flow of data during inference. Dashed lines represent the flow of data during training. Red lines represent the flow of expert data.

When using RL alone MAPS A3C is more difficult to train than SAPS A3C due to the difficulties associated with credit assignment when training using multiple actions (see Introduction). The main problem of training using a SAPS agent however, is that the policy imposes a hard limit on the maximum capability of the agent. This capability is lower than that of an optimal MAPS agent because SAPS policies are a subset of MAPS policies. Indeed, in the best case scenario, a very simple environment where there is no advantage associated with carrying out multiple actions simultaneously, this capability can at best only match that of a MAPS agent. However, the relatively high update frequency of the agent ($\sim$ 15 actions per second) offsets some of the problems associated with single action per time step updates in this game. Running forward whilst strafing can, to a limited extent, be approximated by selecting the forward action in one frame and then the strafe action in the next.

During the early stages of training, the MAPS agent trains more rapidly than the SAPS agent. In the SAPS agent case, firing limits its opportunity to move which in turn adversely affects its ability to pick up boxes and get to the region-of-interest. In the MAPS case, because firing has no effect on locomotion, and allows the agent to hit enemy targets, the agent quickly learns that firing is generally a positive action. However, this initial advantage disappears halfway through training, at which point the SAPS agent learns the benefits of interleaving fire actions and locomotion actions (see red vs blue line in Figure 4). The performance of the SAPS agent eventually surpasses that of the MAPS agent since it is less affected by credit assignment issues.

VII. Analysis

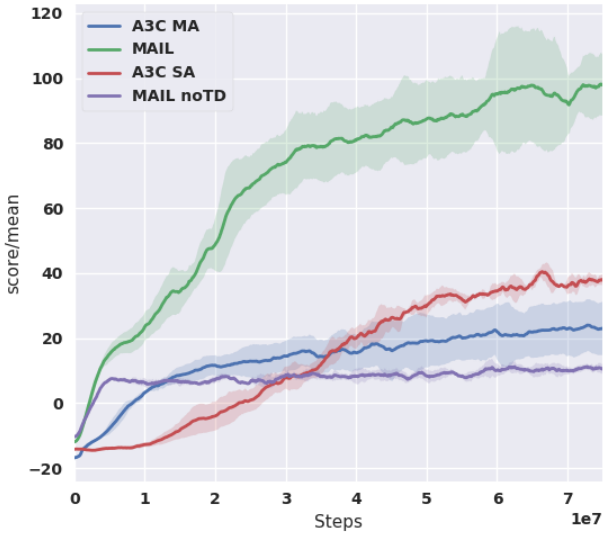


Fig. 4: Multi-action A3C, MAPS (blue). MAIL (green). MAIL without TD RL (purple). Single Action A3C, SAPS (red). The mean and standard deviation over 5 runs are shown for each result.

In this environment, the absolute magnitude of the theoretical performance difference between MAPS A3C and SAPS A3C is difficult to determine because, due to the difficulties of training using TD RL, the SAPS and MAPS agents never reach an optimal policy (see Introduction).

MAIL significantly outperforms both SAPS A3C and MAPS A3C, reaching a final score $\sim$ 2.5 x higher than SAPS A3C and 4x higher than MAPS A3C. It allows an effective policy to be learnt in far fewer steps than when using TD RL alone, exceeding the final score of SAPS A3C after just 17.5M steps, a $\sim$ 4 x reduction in training time (see https://www.youtube.com/watch?v=LW20UbquVBU for example agent behaviour).

This speed-up is most pronounced in the early stages of training when reward sparsity severely limits the effectiveness of TD learning updates; imitation learning provides useful feedback at every training step from the very start of training. Supervised learning allows the vision system to be trained much more rapidly than TD RL. Further, mimicking the behaviour of the expert significantly improves the exploration of state-space in comparison to the unguided random actions in the early stages of TD RL. The MAIL agent quickly learns to collect boxes whilst heading towards the region-of-interest; this behaviour can be seen after less than one hour of training ($<$ 1M steps). From the point of view of the agent, this rapid increase in agent capability significantly reduces reward sparsity and kick-starts the next phase of training, in which temporal difference learning dominates. During this final phase the agent learns to surpass the capabilities of the expert. The mean score of the expert human player was $\sim$ 47; significantly lower than the final score of the MAIL agent, but significantly higher than the other algorithms.

The trained MAIL agent takes full advantage of the MAPS architecture, and typically performs between 1 and 4 actions at once, learning behaviors such as running forward whilst simultaneously moving sideways, turning and shooting. The MAIL agent performs a similar number of actions per step as the expert teacher taking full advantage of the ability to perform multiple actions concurrently. The concurrent action architecture proved critical for effective imitation learning as it was not possible to record high quality expert human data when limiting the expert to performing single actions at a time, in this game.

To better understand how TD RL and IL affect the final MAIL agents capability, we also trained a network without using TD RL (purple curve in Figure 4). The MAPS IL-only network achieved a final score of $\sim$ 15 , significantly lower than all other training runs. This score was achieved after just 5M steps, with no further improvement during the remaining 70M training steps. The results show that, when combined with IL, TD RL has a positive contribution in the very early stages of training; after $\sim$ 2M steps the performance of MAIL surpasses that of pure IL MAIL. At $\sim$ 13M steps the MAIL agents score is twice that of a pure IL agent. It appears that by forcing the network to learn a solution that maximises future reward, TD RL also helps the agent find a more general solution, which allows it to extract more useful information from the expert data; however, testing this hypothesis is left for future work. To asses whether the expert data eventually starts to limit the performance of the agent we compare the performance of a MAIL agent using two different decay rates for the expert data loss (Figure 5). The run using IL data with a higher decay rate reaches a higher final score, suggesting

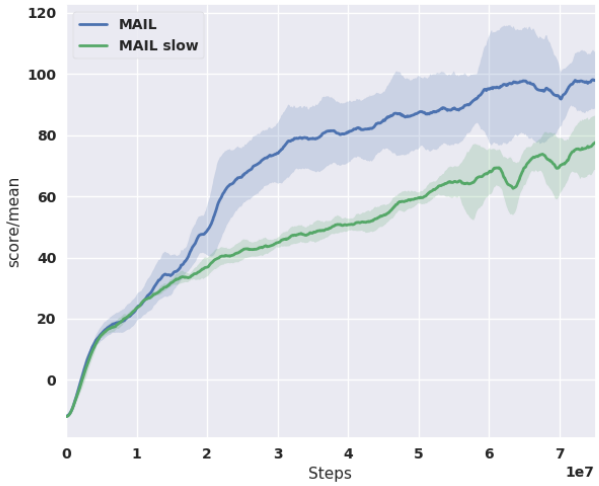


Fig. 5: MAIL (15M) vs MAIL slow decay (50M)

that IL eventually holds back the performance of the agent. These results also seem to indicate that IL learning reduces the variance in agent performance across games, which can be seen in Figure 5.

Interestingly, the behaviour of the trained MAIL agent is distinctly modal in nature. Its behaviour changes significantly depending upon the agents current state. Certain triggers, such as the agent running low on ammo, cause the agent to drastically alter its style of play. These advanced sub-behaviors arise naturally without deliberately partitioning the network to encourage them, i.e. without using concepts such as manager networks. With even more efficient training techniques, deeper networks with simple architectures might be capable of much higher level reasoning than is currently observed. Examples of some of the observed behaviours of the agent include: searching for the waypoint, searching for ammo/health, patrolling the region-of-interest, attacking enemies, fleeing enemies due to low health/ammo, rapidly turning around to face enemies immediately after finding ammo (see Figure 6) and human like navigation around buildings. All these behaviours can be more fully appreciated in the video.

VIII. Future Work

In future work we aim to further enhance the capabilities of the MAIL architecture by adding continuous actions for all rotations. This should provide a number of benefits when combined with the current MAIL architecture. Not only will it provide the agent with more fine grained motor control and reduce the size of the action space, it will also allow much higher quality expert data to be recorded by allowing data to be acquired using a mouse and keyboard or the analogue inputs on a game controller. These improvements should allow the MAIL architecture to be used to train agents to play modern AAA FPS games. Relaxing assumption 1 to more general forms of parametric policies is also left for future work.

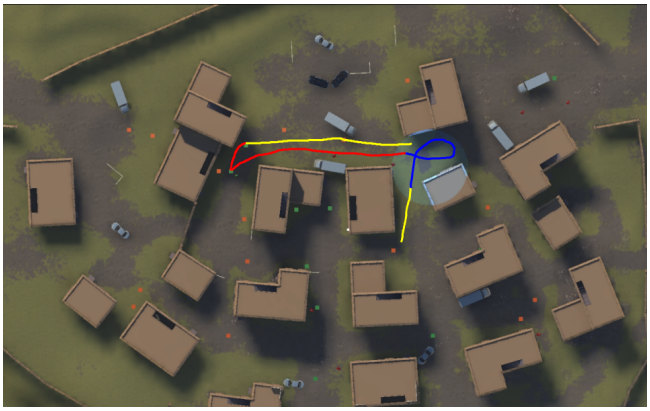


Fig. 6: Example behaviours. Yellow: seeking region of interest, while picking up boxes. Blue: Patrolling the region-of-interest. Red: Seeking Ammo

Acknowledgments

We would like to thank Paul Greveson and Ken Brown for help with game art, Dirk de la Hunt for help with game engine technology and Martin Singh-Blom for insightful discussions.

References

G. Andersen, P. Vrancx, and H. Bou-Ammar. Learning High-level Representations from Demonstrations. *CoRR*, 2018.

M. Babaeizadeh, I. Frosio, S. Tyree, J. Clemons, and J. Kautz. GA3C: GPU-based A3C for Deep Reinforcement Learning. 2016.

M. G. Bellemare, W. Dabney, and R. Munos. A Distributional Perspective on Reinforcement Learning. *CoRR*, 2017.

M. Fortunato, M. G. Azar, B. Piot, J. Menick, I. Osband, A. Graves, V. Mnih, R. Munos, D. Hassabis, O. Pietquin, C. Blundell, and S. Legg. Noisy Networks for Exploration. *CoRR*, 2017.

Y. Gao, Huazhe, Xu, J. Lin, F. Yu, S. Levine, and T. Darrell. Reinforcement Learning from Imperfect Demonstrations. *CoRR*, 2018.

I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. 2016. doi: 10.1016/B978-0-12-391420-0.09987-X.

T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning With A Stochastic Actor. 2018.

T. Hester, M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, D. Horgan, J. Quan, A. Sendonaris, I. Osband, G. Dulac-Arnold, J. Agapiou, J. Z. Leibo, and A. Gruslys. Deep Q-learning from Demonstrations. *CoRR*, 2017a.

T. Hester, M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, A. Sendonaris, G. Dulac-Arnold, G. Net, I. Osband, J. Agapiou, J. Z. Leibo, and A. Gruslys. Learning from Demonstrations for Real World Reinforcement Learning. *CoRR*, 2017b.

B. Kim and J. Pineau. Approximate policy iteration with demonstration data. 2013.

H. M. Le, N. Jiang, A. Agarwal, M. Dudík, Y. Yue, and H. Daumé. Hierarchical Imitation and Reinforcement Learning. *CoRR*, 2018.

T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous Control With Deep Reinforcement Learning. *CoRR*, 2015.

V. Mnih, K. Kavukcuoglu, D. Silver, A. a. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540), 2015. doi: 10.1038/nature14236.

V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous Methods for Deep Reinforcement Learning. 2016.

A. Nair, B. McGrew, M. Andrychowicz, W. Zaremba, and P. Abbeel. Overcoming Exploration in Reinforcement Learning with Demonstrations. *CoRR*, 2017.

A. Y. Ng, D. Harada, and S. Russell. Policy invariance under reward transformations : Theory and application to reward shaping. *Sixteenth International Conference on Machine Learning*, 1999. doi: 10.1.1.48.345.

S. Ross, G. J. Gordon, and J. A. Bagnell. A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning. 2010.

S. Schaal. Learning From Demonstration. 1997.

T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized Experience Replay. *CoRR*, pages 1–23, 2015.

J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. High-Dimensional Continuous Control Using Generalized Advantage Estimation. *CoRR*, 2015.

S. Sharma, A. Suresh, R. Ramesh, and B. Ravindran. Learning to Factor Policies and Action-Value Functions: Factored Action Space Representations for Deep Reinforcement learning. *CoRR*, 2017.

D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

K. Subramanian, G. Tech, C. L. I. Jr, G. Tech, and A. L. Thomaz. Exploration from Demonstration for Interactive Reinforcement Learning. *Aamas*, 2016. ISSN 15582914.

R. S. Sutton and A. G. Barto. Reinforcement Learning : An Introduction. 1998.

H. van Hasselt, A. Guez, and D. Silver. Deep Reinforcement Learning with Double Q-learning. *CoRR*, 2015.

H. Wang and Y. Yu. Exploring Multi-Action Relationship in Reinforcement Learning. 2016.

Z. Wang, N. de Freitas, and M. Lanctot. Dueling Network Architectures for Deep Reinforcement Learning. *arXiv*, 2016.

X. Zhang and H. Ma. Pretraining Deep Actor-Critic Reinforcement Learning Algorithms With Expert Demonstrations. *CoRR*, 2018.

Deep Reinforcement Learning for General Video Game AI

Ruben Rodriguez Torrado* Philip Bontrager* Julian Togelius Jialin Liu
New York University *New York University* *New York University* *Southern University of Science and Technology*
New York, NY New York, NY New York, NY Shenzhen, China
rrt264@nyu.edu philipjb@nyu.edu julian.togelius@nyu.edu liujl@sustc.edu.cn

Diego Perez-Liebana
Queen Mary University of London
London, UK
diego.perez@qmul.ac.uk

***Abstract*—The General Video Game AI (GVGAI) competition and its associated software framework provides a way of benchmarking AI algorithms on a large number of games written in a domain-specific description language. While the competition has seen plenty of interest, it has so far focused on online planning, providing a forward model that allows the use of algorithms such as Monte Carlo Tree Search. In this paper, we describe how we interface GVGAI to the OpenAI Gym environment, a widely used way of connecting agents to reinforcement learning problems. Using this interface, we characterize how widely used implementations of several deep reinforcement learning algorithms fare on a number of GVGAI games. We further analyze the results to provide a first indication of the relative difficulty of these games relative to each other, and relative to those in the Arcade Learning Environment under similar conditions.**

***Index Terms*—deep reinforcement learning, general video game AI, video game description language, OpenAI Gym, advantage actor critic, deep Q-learning**

I. Introduction

The realization that video games are perfect testbeds for artificial intelligence methods have in recent years spread to the whole AI community, in particular since Chess and Go have been effectively conquered, and there is an almost daily flurry of new papers applying AI methods to video games. In particular, the Arcade Learning Environment (ALE), which builds on an emulator for the Atari 2600 games console and contains several dozens of games [1], have been used in numerous published papers since DeepMind's landmark paper showing that Q-learning combined with deep convolutional networks could learn to play many of the ALE games at superhuman level [2].

As an AI benchmark, ALE is limited in the sense that there is only a finite set of games. This is a limitation it has in common with any framework based on existing published games. However, for being able to test the general video game playing ability of an agent, it is necessary to test on games on which the agent was not optimized. For this, we need to be able to easily create new games, either manually or automatically, and add new games to the framework. Being able to create new games easily also allows the creating of games made to test particular AI capacities.

The General Video Game AI (GVGAI) competitions and framework were created with the express purpose of providing a versatile general AI benchmark [3], [4], [5], [6]. The *planning tracks* of the competition, where agents are given a forward model allowing them to plan but no training time between games, have been very popular and seen a number of strong agents based on tree search or evolutionary planning submitted. A *learning track* of the competition has run once, but not seen many strong agents, possibly because of infrastructure issues. For the purposes of testing machine learning agents (as opposed to planning agents), GVGAI has therefore been inferior to ALE and similar frameworks.

In this paper, we attempt to rectify this by presenting a new infrastructure for connecting GVGAI to machine learning agents. We connect the framework via the OpenAI Gym interface, which allows the interfacing of a large number of existing reinforcement learning algorithm implementations. We plan to use this structure for the learning track of the GVGAI competition in the future. In order to facilitate the development and testing of new algorithms, we also provide benchmark results of three important deep reinforcement learning algorithms over eight dissimilar GVGAI games.

II. Background

A. General Video Game AI

The General Video Game AI (GVGAI) framework is a Java-based benchmark for General Video Game Playing (GVGP) in 2-dimensional arcade-like games [5]. This framework offers a common interface for bots (or agents, or controllers) and humans to play any of the more than 160 single- and two-player games from the benchmark. These games are defined in the Video Game Description Language (VGDL), which was initially proposed by Ebner *et al.* [3] at the Dagstuhl Seminar on Artificial and Computational Intelligence in Games.

VGDL [7] is a game description language that defines 2-dimensional games by means of two files, which describe the

978-1-5386-4359-4/18/$31.00 ©2018 IEEE

game and the level respectively. The former is structured in four different sections, detailing game sprites present in the game (and their behaviors and parameters), the interactions between them, the termination conditions of the game and the mapping from sprites to characters used in the level description file. The latter describes a grid and the sprite locations at the beginning of the game. These files are typically not provided to the AI agents, who must learn to play the game via simulations or repetitions. More about VGDL and sample files can be found on the GVGAI GitHub project[1].

The agents implement two methods to interact with the game: a constructor where the controller may initialize any structures needed to play, and an `act` method, which is called every game frame and must return an action to execute at that game cycle. As games are played in real-time, the agents must reply within a time budget (in the competition settings, 1 second for the constructor and 40ms in the `act` method) not to suffer any penalty. Both methods provide the agent with some information about the current state of the game, such as its status (if it is finished or still running), the player state (health points, position, orientation, resources collected) and anonymized information about other sprites in the game (so their types and behaviours are not disclosed). Additionally, controllers also receive a forward model (in the *planning* setting) and a screen-shot of the current game state (in the *learning* setting).

The GVGAI framework has been used in a yearly competition, started in 2014, and organized around several tracks. Between the single- [4] and the two-player [8] GVGAI planning competitions, more than 200 controllers have been submitted by different participants, in which agents have to play in sets of 10 unknown games to decide a winner. These tracks are complemented with newer ones for single-player agent learning [9], [6], level [10] and rule generation [11]. Beyond the competitions, many researchers have used this framework for different types of work on agent AI, procedural content generation, automatic game design and deep reinforcement learning, among others [6].

In terms of learning, several approaches have been made before the single-player learning track of the GVGAI competition was launched. The first approach was proposed by Samothrakis *et al.* [12], who implemented Separable Natural Evolution Strategies (S-NES) to evolve a state value function in order to learn how to maximize victory rate and score in 10 games of the framework. Samothrakis *et al.* [12] compared a linear function approximator and a neural network, and two different policies, using features from the game state.

Later, Braylan and Miikkulainen [13] used logistic regression to learn a forward model on 30 games of the framework. The objective was to learn the state (or, rather, a simplification consistent of the most relevant features of the full game state) that would follow a previous one when an action was supplied, and then apply this model in different games, assuming that some core mechanics would be shared among the different games of the benchmark. Their results showed that these learned object models improved exploration and performance in other games.

More recently, Kunanusont *et al.* [14] interfaced the GVGAI framework with DL4J[2] in order to develop agents that would learn how to play several games via screen capture. 7 games were employed in this study, of increasing complexity and screen size and also including both deterministic and stochastic games. Kunanusont *et al.* [14] implemented a Deep Q-Network for an agent that was able to increase winning rate and score in several consecutive episodes.

The first (and to date, only) edition of the single-player learning competition, held in the IEEE's 2017 Conference on Computational Intelligence in Games (CIG2017), received few and simple agents. Most of them are greedy methods or based on Q-Learning and State-Action-Reward-State-Action (SARSA), using features extracted from the game state. For more information about these, including the final results of the competition, the reader is referred to [6].

B. Deep Reinforcement Learning

A Reinforcement Learning (RL) agent learns through trial-and-error interactions with a dynamic environment [15] and balance the reward trade-off between long-term and short-term planning. RL methods have been widely studied in many disciplines, such as operational research, simulation-based optimization, evolutionary computation and multi-agent system, including games. The cooperation between the RL methods and Deep Learning (DL) has led to successful applications in games. More about the work on Deep Reinforcement Learning till 2015 can be found in the review by J. Schmidhuber [16]. For instance, Deep Q-Networks has been combined with RL to play several Atari 2600 games with video as input [17], [2]. Vezhnevets *et al.*[18] proposed STRategic Attentive Writer-exploiter(STRAWe) for learning macro-actions and achieved significant improvements on some Atari 2600 games. *AlphaGo*, combined tree search with deep neural networks to play the game of Go and self-enhanced by self-playing, is ranked as 9 dan professional [19] and is the first to beat human world champion of Go. Its advanced version, *AlphaGo Zero* [20] is able to learn only by self-playing (without the data of matches played by human players) and outperforms *AlphaGo*.

During the last few years, several authors have improved the results and stability obtained with the original Deep Q-Networks paper. Wang et. al. [21] introduces a new architecture for the networks know as dueling network, this new architecture uses two separate estimators: one for the state value function and one for the state-dependent action advantage function. The main benefit of this factoring is to generalize learning across actions without imposing any change to the underlying reinforcement learning algorithm.

Mnih et. al., in 2016, successfully applied neural networks to actor-critic RL [22]. The network is trained to predict both

[1]https://github.com/EssexUniversityMCTS/gvgai/wiki/VGDL-Language

[2]Deep Learning for Java: https://deeplearning4j.org/

a policy function and a value function for a state, the actor and the critic. Asynchronous Advantage Actor-Critic, A3C, is inherently parallelizable and allows for a big speedup in computation time. The interaction between the policy output and the value estimates has been shown to be relatively stable and accurate for neural networks. This new approach increases the score obtained from the original DQN paper, reducing the computational time by half even without using CPU.

C. OpenAI Gym

RL is a hot topic for the research community of artificial intelligence. Recent advances that combine DL with RL (Deep Reinforcement Learning) have shown that model-free optimization, or policy gradients, can be used for complex environments. However, in order to continue testing new ideas and increasing the quality of results, the research community needs good benchmark platforms to compare results. This is the main goal of OpenAI GYM platform [23].

The OpenAI GYM platform provides a high variety of benchmark, such as Arcade Learning Environment (ALE) [24], which is a collection of Atari 2600 video games. OpenAI Gym has more environments for testing RL in different types of environments. For example, MuJoCo is used to test humanoid like movement in 2D and 3D.

III. Methods

While one of the main benefits for GVGAI is the ease to which new games can be created for a specific problem, we also feel it is necessary to place the current GVGAI games in the context of other existing environments. This serves two purposes: it further demonstrates the strengths and weaknesses of the current generation of reinforcement learning algorithms, and it allows results achieved on GVGAI to be compared to other existing environments.

A. GVGAI-OpenAI embedding

The learning competition is based on the GVGAI framework, but no forward model is provided to the agents, thus no simulations of a game are accessible. However, an agent still has access to the observation of current game state, a *StateObservation* object, provided as a Json object in *String* or as a screen-shot of the current game screen (without the screen border) in *png* format. At every game tick, the server sends a new game state observation to the agent, the agent returns either an action to play in 40ms or requests to abort the current game. When a game is finished or aborted, the agent can select the next level to play, among the existing levels (usually 5 levels). This setting makes it possible to embed the GVGAI framework as an OpenAI Gym so that the reinforcement learning algorithms can be applied to learn to play the GVGAI games. Thanks to VGDL, it is easy to design and add new games and levels to the GVGAI framework.

The main framework is described in the manual by Liu *et al.* [9], as well as the default rules in the framework. Only 5 minutes is allowed to each of the agents for learning. It is notable that only the decision time (no more than $40ms$ per game tick) used by the agent is included, while the game advancing time, game state serialization time and communication time between the client and agent are not included. The real execution of the learning phase can last several hours.

B. GVGAI Games

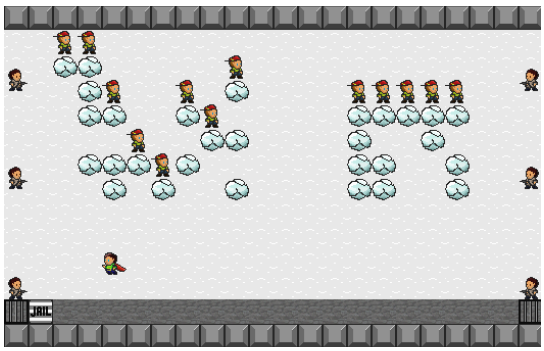


Figure 1: Screenshot of game *Superman*. In this game, innocent civilians are standing on clouds while malicious actors spawn around the edge of the screen and attempt to shoot the clouds out from underneath them. If all the clouds are gone the civilian will fall and only Superman can save them by catching them for 1 point. Superman can also jail the villains for 1 point. If Superman catches all the villains, the player wins and earns an additional 1000 points.

The GVGAI environment currently has over 160 games and counting. To showcase the environment and the challenges that already exist we sample a number of games to benchmark against popular reinforcement learning algorithms.

Our criteria for sampling games was informal but based on several considerations. Since many of the games in the GVGAI framework have been benchmarked with planning agents, we can roughly rank the games based on how difficult these games are for planning. We tried to get an even distribution across the range going from games that are easy for planning agents, like Aliens, to very difficult, like Superman. The game difficulties are based on the analysis by Bontrager *et al.* [25]. Other things we considered were having a few games that also exist in Atari for some comparison and including games that we believed would provide interesting challenges to reinforcement learning agents. Some games in VGDL contain stochastic components as well, mostly in the form of NPC movement. GVGAI has five levels for each game, we used the first level for each game for all the training.

We settled on Aliens, Seaquest, Missile Command, Boulder Dash, Frogs, Zelda, Wait For Breakfast, and Superman. The first five mentioned are modeled after their similarly named Atari counterpart. Zelda consists of finding a target while killing or avoiding enemies. Frogs is modeled after Frogger which is also similar to the Atari Freeway game. Wait For Breakfast (Figure 2) is a strange game where the player must go to a breakfast table where food is being served a sit there for a short amount of time. This is not usually what people

think of as a game but provides an interesting challenge for bots. Finally, Superman (Figure 1) is a complicated game that involves saving people in a dangerous environment with no reward until the person is safe. A full version of our implementation can be found on GVGAI GYM repository [3].

Figure 2: Screenshot of game *Wait For Breakfast*. In this game, all tables are empty when a game starts. At a randomly selected game tick, a waiter (in black) serves a breakfast to the table with only one chair. The player (in green) wins the game only if it sits on the chair on the table after the breakfast is served and eats it. The player loses the game if it leaves the chair once breakfast has been served without eating it.

C. Benchmarks

To have standardized results we decided to choose a few popular reinforcement learning algorithms that are provided by the OpenAI Gym baselines library. The baselines are open implementations of these algorithms and are closely based on the original papers [26]. The hope is that by using publicly vetted and accessible code that our results will be comparable to other work and reproducible.

From OpenAI's baseline library we selected three algorithms: Deep Q-Networks (DQN), Prioritized Dueling DQNs, and Advantage Actor-Critic (A2C). These were chosen in part because they have been well documented in similar environments such as ALE. DQN and A3C, which A2C is based on, are the baseline for which many new RL developments are scored against. For this reason, we felt it made sense to use these to benchmark the GVGAI games.

For all three baselines, we used the same network first described in Mnih *et al.* for playing Atari [17]. This consists of 3 convolutional layers and two fully connected layers as seen in Table I. GVGAI is providing screen-shots for each game state that the convolutional network learns to interpret. Each algorithm is trained on one million frames of a particular game. From initial testing, it appeared that one million calls were enough to give an indication of the difficulty of a game for our agents while also being realistic in terms of computational resources. It is also a step in the right direction for the learning track of GVGAI where there are very tight time constraints. To accommodate the smaller number of training iterations, we changed a few training parameters. Buffer size, the size of replay memory, was set to 50,000,

[3]https://github.com/rubenrtorrado/GVGAI_GYM

Layer Type	Layer Parameters		
	Depth	Kernel	Stride
Convolution 1	32	8	4
Convolution 2	64	4	2
Convolution 3	64	3	1
Fully Connected	256		
Fully Connected	Action Space		

Table I: This table represents the architecture of the network used to play each game. For convolutional layers, depth refers to the convolutional filters and for the fully connected layers it refers to the output size.

the network starts learning after only 1000 initial decisions, and the target Q-network gets updated every 500 steps.

We test both the original DQN and a modified DQN. OpenAI Baselines has a DQN implementation that is based on the original DQN but it also offers prioritized experience replay and dueling networks as options that can be turned on since they work together with the original implementation [26]. We tested the original for comparisons and also ran DQN with the two additional modifications to get results from a more state of the art DQN. We used the baseline defaults for the network with a couple of exceptions pertaining to training time. The defaults have been tuned for ALE and should carry over.

To test A3C, OpenAI provides A2C. This is a synchronous version that they found to be more efficient and perform just as well on Atari [26]. This was also tested with the baseline defaults with the same changes made for DQN. Each baseline was tested on every game for one million calls, resulting in a total of 24 million calls.

IV. Results and Discussion

Here we present the results of training the baselines on each game. The results show the performance of the provided baselines for a sample of the games in the GVGAI framework. This provides insight into how the baselines compare to other AI techniques and to how the GVGAI environment compares to other environments.

Finally, this section is structured in three parts. First, the results of training the learning algorithms on the games are provided with some additional qualitative remarks. Second, the GVGAI environment is compared to the Atari environment. Third, the reinforcement agents are compared to planning agents that have been used within the framework.

A. Results of learning algorithms

Figure 3 shows the training curves for DQN (red), Dueling Prioritized DQN (blue) and A2C (green). The graphs show the total rewards for playing up to that point in time. Rewards are completely defined by the game description so they can't be compared between different games. This is done by reporting the sum of the incremental rewards for the episode at a given time step. Since this data is noisy due to episode restarts, the 20 results are averaged to smooth the graph and better show a trend. A2C allows running in parallel, we were able to run 12 networks in parallel at once. To keep the comparisons

fair, A2C is still only allowed one million GVGAI calls and therefore each of the 12 networks is given one-twelfth of a million calls each. This results in the training graph seen in Figure 4. To compare this with the linear algorithms, each time step of A2C is associated with 12 time-steps of the DQN algorithms in Figure 3. The value for each time step of A2C is the average of all 12 rewards.

Due to the fact that we are running experiments on different machines with different GPU and CPU configurations, we align the results on iterations instead of time. It is important to note that since A2C runs its fixed number of GVGAI calls in parallel, it runs at about 5x the speed of DQN on a machine with two NVIDIA Tesla k80 GPUs.

Figure 4 shows the training curve in parallel for A2C on Boulder Dash. The individual agents are chaotic which helps A2C break out of local minima. This also points to the importance of the exploration algorithm in learning to play games. In Boulder Dash, as long as one of the 12 workers found an improvement they would all gain.

The agents were able to learn on most of the games that were sampled. A2C performed the best for most of the games tested. Though it's important to remember a relatively small computational budget was allowed for these algorithms and the others might eventually catch up. 8 games is also a small sample for comparing which algorithm is the best. A2C seems to benefit from sampling more initial conditions and starts with a higher score.

DQN and Prioritized Dueling DQN were both given the same initial seed so they had the same initial exploration pattern. For this reason, both algorithms tended to start out with similar performance and then diverge as time goes on. Prioritized Dueling DQN seems to slightly outperform vanilla DQN, but on overall they are very similar. A2C could not be compared in this way as it intentionally is running different explorations in parallel and then learn from all of them at the same time. This can explain why A2C tends to start out better right from the beginning, especially in Aliens. It is benefiting from 12 different initial conditions in this case.

Available rewards have a big impact on the success of RL and that is not different in the GVGAI environment. The games where the agents performed worst were the games that had the least feedback. For this work, we left the games in their current form, but it is very easy for researchers to edit the VGDL file and modify the reward structure to create various experiments.

The games sampled here vary a lot in terms of the rewards they offer. Frogs and Wait For Breakfast only provide a single point for winning. This is evident in their training graphs. For Frogs, none of the agents appear to have found a winning solution in the calls allotted. This resulted in a situation where RL could not play the game. Wait For Breakfast has a simpler win condition in a very static environment. The agent had to flounder around a lot until it bumped into the correct location for a few consecutive iterations. The environment is very static so once a solution is found it just has to memorize it. A2C has the exploration advantage and can find the solution sooner but it keeps exploring and does not converge to the single conclusion as quickly.

Missile Command shows a similar performance for the three algorithms. Although Prioritized Dueling DQN finds a higher value in earlier stages, The three algorithms get trapped in a local optimum. In the game missile command, four fire-balls target three bases. To get all 8 points the player has to defend all three. One of the bases gets attacked by two fire-balls which make it hard to defend. To have time to save the third base requires very accurate play, the agents did not seem to be able to maintain a perfect score because a few missteps led to 5 points. The reward plain is very non-linear for this game.

Superman takes this difficulty to the next level. The game is very dynamic with many NPCs modifying the environment in a stochastic manner. This means that any actions that the agent takes will have a big impact on the environment in the future. On top of this, the way to get the most points is to capture the antagonists and take them to jail. No points are awarded for capture, only for delivery to jail. This introduces a delayed reward which is a barrier to discovery. Knowing this, the results from the training on this game make sense. The agents were occasionally able to stumble on a good pattern but they could not reproduce the success in the stochastic environment.

DQN and Prioritized Dueling DQN struggled to play Boulder Dash. In Boulder Dash, when the player collects a diamond for points, a rock falls toward them. This means there is negative feedback if an agent collects a diamond and doesn't move. Not collecting any diamonds and surviving appears to be an obvious local optimum that the agents have a hard time escaping. On the other hand, A2C was able to discover how to collect diamonds and survive, with a clear trend of continuing to improving.

Seaquest is a good example of a game that is not too hard but has a lot of random elements. The agent can get a high score if it can survive the randomly positioned fish, catch the randomly moving diver, and take it to the surface. This requires the agent to learn to chase the diver which none of the agents appear to be doing. The high noise in the results is most likely from the agents failing to learn the general rules behind the stochasticity. Additionally, the player needs to go to the surface every 25 game ticks or it loses the game, which may be something hard to learn for the agents.

Finally, Zelda is a fairly good game for reinforcement learning. Though, the game is not too similar to its namesake. The player must find a key and use it to unlock the exit while fighting enemies. Each event provides feedback which allows the agents to learn the game well.

B. Comparison with ALE

Reinforcement learning research has been making a lot of progress on game playing in the last few years and the benchmark environments need to keep up. ALE is a popular 2D environment. It consists of a reasonably large set of real games and all the games have been designed for humans. Yet, the game set is static and cannot provide new challenges as

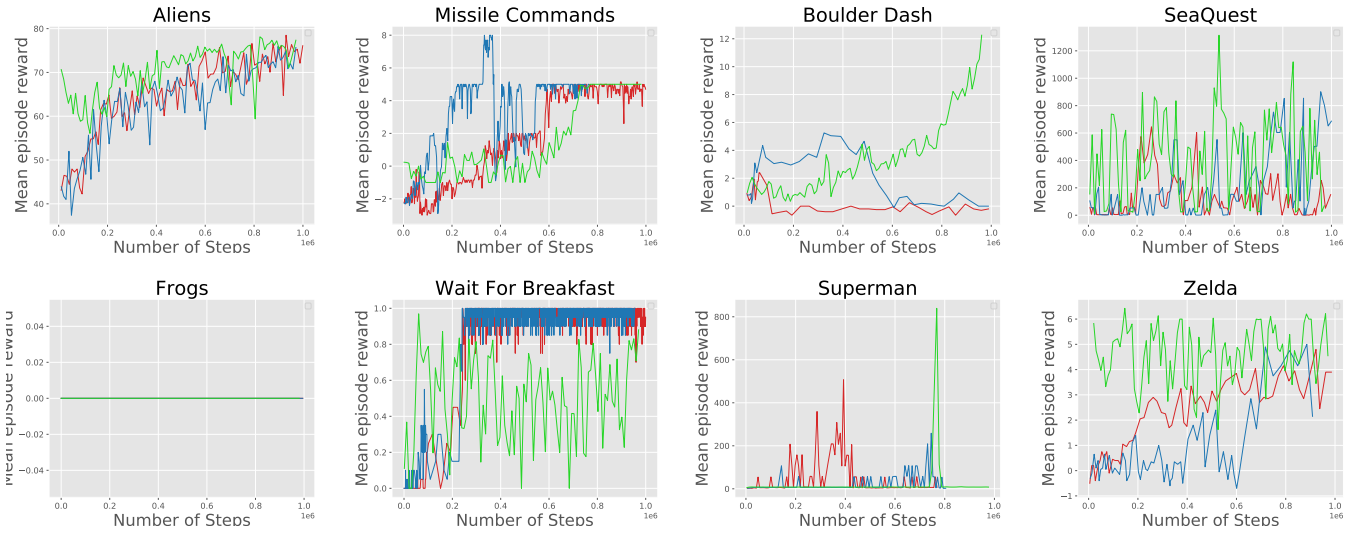


Figure 3: Training reward for DQN (red), Prioritized Dueling DQN (blue), and A2C (green). The reward is reported on the y-axis and is different for each game. As an example, Frogs only returns a score of 1 for winning and 0 otherwise. Each algorithm is trained on one million game frames.

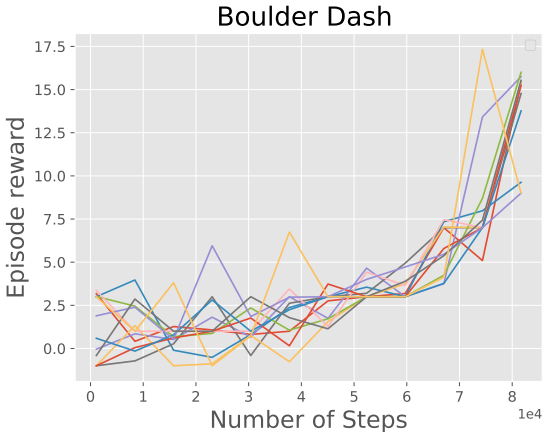


Figure 4: Training reward for all 12 workers of A2C learning on Boulder Dash

researchers experiment with the strengths and weaknesses of different algorithms.

GVGAI currently has over twice the number of games as ALE and with active research more are added every year. The VGDL language also makes it possible for researchers to design new games. Truly stochastic games can be designed and multiple levels can be included to test how well an algorithm can generalize. The VGDL engine also provides a forward model that can be incorporated in the future to allow hybrid algorithms to learn and plan.

While these games allow targeted testing of AIs, they tend to not be designed with humans in mind and can be hard to play. Readers are also not as familiar with the games as they are in Atari and therefore might lack some of the intuition. Another drawback is speed. The engine is written in Java and communicating through a local port to Python. While still very fast, training will run a few times slower than Atari. Currently, there is ongoing development to optimize the communication between the two languages.

While both environments share some games, the performance on these games cannot be compared directly. GVGAI has games that are inspired by Atari but they are not perfect replicas and the author of the VGDL file can decide how close to match the original and how to handle score. Yet, looking at similar games in both environments seems to show that GVGAI can have many of the characteristics of Atari: such as fairly good performance on Aliens and poor performance on Seaquest.

The ALE has done a lot for providing a standard benchmark for new algorithms to be tested against. GVGAI is more fluid and changing but it allows researchers to constantly challenge the perceived success of new RL agents. The challenges for computers can advance with them all the way to general video game playing. On top of that, we provide the results here to propose that doing well on GVGAI is at least comparable doing well on ALE and we show that there are games on GVGAI that still are not beaten.

C. Comparison with planning algorithms

In order to compare the performance of our learning algorithms with the state-of-art, we have used the results obtained in [25]. This paper explores clustering GVGAI games to better understand the capabilities of each algorithm and subsequently use several agents to test the performance of each representative game. The tested agents may be classified in Genetic Algorithms (GA), Monte Carlo Tree Search (MCTS), Iterative With and Random Sample (RS). To compare results, we took the agent with the high score for each category in a target environment.

In Table II we compare the performance of the reinforcement-learned neural network agents with high-performing planning agents. This is very much a case of comparing apples and oranges: the learning-based agents have been trained for hours for the individual game it is being tested on whereas the planning-based agents have had no training time whatsoever and are supposed to be ready to play any game at any point, and the planning-based agents have access to a forward model which the learning agent does not. In other words, each type of agent has a major advantage over the other, and it is a priori very hard to say which advantage will prove to be the most important. This is why this comparison is so interesting.

Beginning with Aliens, we see that all agents learn to play this game well. This is not overly surprising, as all Non-player Characters (NPC) and projectiles in this game behave deterministically (enemy projectiles are fired stochastically, but always takes some time to reach the player) and the game can be played well with very little planning; the main tasks are avoiding incoming projectiles and firing at the right time to hit the enemy. The former task can be solved with a reactive policy, and the latter with a minimum of planning and probably also reactively.

Wait for Breakfast was solved perfectly by all agents except the standard MCTS agent, which solved it occasionally. This game is easily solved if you plan far enough ahead, but it is also very easy to find a fixed strategy for winning. It punishes "jittery" agents that explore without planning.

Frogs is only won by the planning agents (GA and IW always win it, MCTS sometimes wins it) whereas it is never won by the learning algorithm. The simple explanation for this is that there are no intermediate rewards in Frogs; the only reward is for reaching the goal. There is, therefore, no gradient to ascend for the reinforcement learning algorithms. For the planning algorithms, on the other hand, it is just a matter of planning far enough ahead. (Some planning algorithms do better than others, for example, Iterative Width looks for intermediate states where facts about the world have changed.) The reason why learning algorithms perform well on Freeway, the Atari 2600 clone of Frogger, is that it has plenty of intermediate rewards - the player gets a score for advancing each lane.

Two of the planning agents and all three learning agents perform well on Missile Command; there seems to be no meaningful performance difference between the best planning algorithms (IW) and the learning agents. It seems possible to play this game by simply moving close to the nearest approaching missiles and attacking it. What is not clear is why MCTS is performing so badly.

Seaquest is a relatively complex game requiring both shooting enemies, rescuing divers and managing oxygen supply. All agents play this game reasonably well, but somewhat surprisingly, the learning agents perform best overall and A2C is the clear winner. The presence of intermediate rewards should work in the learning agents' favor; apparently, the learning agents easily learn the non-trivial sequence of tasks as well.

Boulder Dash is perhaps the most complex game in the set. The game requires both quick reactions for the twitch-based gameplay of avoiding falling boulders and long-term planning of in which order to dig dirt and collect diamonds so as not to get trapped among boulders. Here we have the interesting situations the one planning algorithm (MCTS) and one learning algorithm (A2C) plays the game reasonably well, whereas the other algorithms (both planning and learning) perform much worse. For the planning algorithms, the likely explanation is that GA has too short planning horizon and IW does not handle the stochastic nature of the enemies.

For Zelda, which combines fighting random-moving enemies and finding paths to keys and doors (medium-term planning), all agents performed comparably. The tree search algorithms outperformed the GA, and also seem to outperform the learning agents, but not by a great margin.

V. Conclusion

In this paper, we have created a new reinforcement learning challenge out of the General Video Game AI Framework by connecting it to OpenAI Gym environment. We have used this setup to produce the first results of state-of-art deep RL algorithms on GVGAI games. Specifically, we tested DQN, Prioritized Dueling DQN and Advance Actor-Critic (A2C) on eighth representative GVGAI games.

Our results show that the performance of learning algorithm differs drastically between games. In several games, all the tested RL algorithms can learn good stable policies, possibly due to features such as memory replay and parallel actor-learners for DQN and A2C respectively. A2C reaches a higher score than DQN and PDDQN for 6 of the 8 environments tested without memory replay. Also, when trained on the GVGAI domain using 12 CPU cores, A2C trains five times faster than DQN trained on a Tesla Nvidia GPU.

But there are also many cases where some or all of the learning algorithms fail. In particular, DQNs and A2C perform badly on games with a binary score (win or lose, no intermediate rewards) such as Frogs. Also, we observed a high dependency of the initial conditions which suggests that running multiple times is necessary for accurately benchmarking DQN algorithms. Finally, some complex games (e.g. Seaquest) show problems of stabilization when we are training with default parameters of OpenAI baselines. This reflects that a modification of replay memory or the schedule of the learning rate parameters are necessary to improve convergence in several environments.

We also compared learning agents (which have time for learning but not a forward model) with planning agents (which get no learning time, but do get a forward model). The results indicate that in general, the planning agents have a slight advantage, though there are large variations between games. The planning agents seem better equipped to deal with making decisions with a long time dependency and no intermediate rewards, but the learning agents performed better on e.g.

Games	Random Agent	Planning Agents			Learning Agents		
		Genetic Algorithm	Monte Carlo Tree Search	Iterative Width	DQN	Prioritized Dueling DQN	A2C
Aliens	52	**80.4**	72.6	80.2	75	74	77
Wait For Breakfast	0	**1**	0.4	**1**	**1**	**1**	**1**
Frogs	-2	**1**	-0.4	**1**	0	0	0
Missile Command	-2.2	2.6	-3	6.8	5	**8**	5
Seaquest	17.2	435	638.2	224.6	600	800	**1200**
Boulder Dash	1.4	3.4	**16.4**	8.8	2.5	5	15.5
Zelda	-5.2	3.4	6.8	**7.6**	4.2	4.2	6
Superman	4	157	**6699**	130.2	500	0	800

Table II: Learning score comparison of learning algorithms (DQN, Prioritized Dueling DQN and A2C) with random and planning algorithms (Genetic Algorithms, MCTS and Iterative Width). The results of planning and random are taken from [25] and correspond to the best performing instance of each algorithm.

Seaquest (a complex game) and Missile Command (a simple game).

As researchers experiment with more the existing games, design specific games for experiments, and participate in the competition, we expect to gain new insights into the nature of various learning algorithms. There is an opportunity for new games to be created by humans and AIs in an arms race against improvements from game-playing agents. We believe this platform can be instrumental to scientifically evaluating how different algorithms can learn and evolve to understand many changing environments.

ACKNOWLEDGEMENT

This work was supported by the Ministry of Science and Technology of China (2017YFC0804003).

(*) The first two authors contributed equally to this work.

REFERENCES

[1] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents." *J. Artif. Intell. Res.(JAIR)*, vol. 47, pp. 253–279, 2013.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[3] M. Ebner, J. Levine, S. M. Lucas, T. Schaul, T. Thompson, and J. Togelius, "Towards a video game description language," in *Dagstuhl Follow-Ups*, vol. 6. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2013.

[4] D. Perez-Liebana, S. Samothrakis, J. Togelius, T. Schaul, S. M. Lucas, A. Couëtoux, J. Lee, C.-U. Lim, and T. Thompson, "The 2014 general video game playing competition," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 8, no. 3, pp. 229–243, 2016.

[5] D. Perez-Liebana, S. Samothrakis, J. Togelius, S. M. Lucas, and T. Schaul, "General Video Game AI: Competition, Challenges and Opportunities," in *Thirtieth AAAI Conference on Artificial Intelligence*, 2016, pp. 4335–4337.

[6] D. Perez-Liebana, J. Liu, A. Khalifa, R. D. Gaina, J. Togelius, and S. M. Lucas, "General video game ai: a multi-track framework for evaluating agents, games and content generation algorithms," *arXiv preprint arXiv:1802.10363*, 2018.

[7] T. Schaul, "A video game description language for model-based or interactive learning," in *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*. IEEE, 2013, pp. 1–8.

[8] R. D. Gaina, A. Couetoux, D. J. N. J. Soemers, M. H. M. Winands, T. Vodopivec, F. Kirchgeβner, J. Liu, S. M. Lucas, and D. Perez-Liebana, "The 2016 two-player GVGAI competition," *IEEE Transactions on Computational Intelligence and AI in Games*, 2017.

[9] J. Liu, D. Perez-Liebana, and S. M. Lucas, "The single-player GVGAI learning framework - technical manual," 2017. [Online]. Available: http://www.liujialin.tech/publications/GVGAISingleLearning_manual.pdf

[10] A. Khalifa, D. Perez-Liebana, S. M. Lucas, and J. Togelius, "General video game level generation," in *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference*. ACM, 2016, pp. 253–259.

[11] A. Khalifa, M. C. Green, D. Pérez-Liébana, and J. Togelius, "General Video Game Rule Generation," in *2017 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2017.

[12] S. Samothrakis, D. Perez-Liebana, S. M. Lucas, and M. Fasli, "Neuroevolution for general video game playing," in *2015 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2015, pp. 200–207.

[13] A. Braylan and R. Miikkulainen, "Object-model transfer in the general video game domain," in *Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2016.

[14] K. Kunanusont, S. M. Lucas, and D. Pérez-Liébana, "General Video Game AI: Learning from Screen Capture," in *2017 IEEE Conference on Evolutionary Computation (CEC)*. IEEE, 2017.

[15] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998, vol. 1, no. 1.

[16] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural networks*, vol. 61, pp. 85–117, 2015.

[17] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[18] A. Vezhnevets, V. Mnih, S. Osindero, A. Graves, O. Vinyals, J. Agapiou *et al.*, "Strategic attentive writer for learning macro-actions," in *Advances in neural information processing systems*, 2016, pp. 3486–3494.

[19] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[20] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[21] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, "Dueling network architectures for deep reinforcement learning," in *International Conference on Machine Learning*, 2016, pp. 1995–2003.

[22] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International Conference on Machine Learning*, 2016, pp. 1928–1937.

[23] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[24] J. V. M. G. Bellemare, Y. Naddaf and M. Bowling., "The arcade learning environment: An evaluation platform for general agents," *J. Artif. Intell. Res*.

[25] P. Bontrager, A. Khalifa, A. Mendes, and J. Togelius, "Matching games and algorithms for general video game playing," in *Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2016, pp. 122–128.

[26] P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, "Openai baselines," https://github.com/openai/baselines, 2017.

Promotion of Learning Motivation through Individualization of Learner-Game Interaction

Dipl.-Päd. Sandra Kaczmarek
Chair of Enterprise Logistics
TU Dortmund University
Dortmund, Germany
kaczmarek@lfo.tu-dortmund.de

M.sc.ing. Sintija Petroviča
Department of Artificial Intelligence and Systems Engineering
Riga Technical University
Riga, Latvia
sintija.petrovica@rtu.lv

***Abstract*—Educational games have been used in educational settings for ages. The reasons behind this interest are different, e.g., games increase enjoyment, involvement, and motivation, as well as they influence emotions. In an area where competence level and challenge level are well balanced the so-called "flow-experience" appears. But how can we maintain "flow-experience" by using serious games? Computational intelligence could be a pioneer answer, if the technology will be implemented in a didactic and meaningful manner. A serious game with emotion-based adaptation has been developed and described in the paper. An experimental study with 244 participants has been carried out involving the usage of the developed game. Results of the experiment in terms of learning outcomes, flow-experience, and motivation are described in the paper.**

***Index Terms*—educational games, learning motivation, flow, computational intelligence, learner-game interaction, experimental design**

I. Introduction

Serious games are (computer) games that contain principles and mechanisms, which demonstrably influence the learning process and motivation in a positive manner [1, 2, 3]. In contrast to linear training formats, serious games provide a framework that supports the individual learning process by offering the possibility to acquire knowledge individually and exploratively and by using this knowledge in practice to gain experience. Games are considered to be beneficial for learning because they incorporate two fundamental aspects [4]: (1) educational aspect related to the learning content and strategies presented to learners and (2) playful aspect that allows players to act, explore, take rewards, etc. But the fact that learning in an occupational context is still perceived as hard work often hinders these potentials [5]. In addition, a lack of theoretical evidence for the successful use of serious games within companies exists and the clarification for whether and how they develop their full potential in operational practice is still missing [6, 7, 8]. A fundamental lack of empirical academic research is a point of criticism that researchers in [9, 1, 2] refer to.

For this reason, TU Dortmund conducted a comprehensive qualitative requirements analysis for the development and use of serious games. Based on the collaborative research methodology, according to Hever, requirements for serious games from the point of view of game developers, pedagogues, experts and future users were raised exemplarily for the domain of logistics with the help of a mix of methods of qualitative and action research. These were mirrored, aggregated and condensed into generalizable requirements using findings from a comprehensive literature review of studies, theories and methodological approaches [3, 10].

The central insight is that a game is only as good as the user experiences it. Therefore, a user-centered game design approach is a crucial criterion for developing a valuable learning experience in serious gameplay. If a fundamental knowledge about skills, competences and previous experiences of the learner is gathered, the game developers are able to provide needs-based information, support and stumbling blocks at the right moments in the game situation. The direct feedback of the players' actions in the game situation is one of the most effective and revealing advantages of serious games. The fact that the learner always has insight into his or her abilities accompanies the learner on the way to self-knowledge about his or her own strengths and weaknesses [3].

All these requirements relate to the individual needs of learners during the learning process with the serious game. So, it is all about learner-game interaction and its individualization. On the one hand, a comprehensive requirements analysis and continuous involvement of future users in the game design process can help to understand needs of users and transfer them into the game design. On the other hand, there is a problem that all contingencies have to be taken into account during the development process in order to ensure a certain flexibility during the game situation. In particular, new technologies such as computational intelligence offer a great potential to guarantee this necessary flexibility for feedback and individual support of the learner according to his/her needs and situation, while at the same time reducing the development effort. Therefore, TU Dortmund and Riga Technical University (RTU) pool their competencies and work together to find out how computational intelligence can individualize the learner-game interaction and how this affects the learning processes and motivation in a meaningful didactic manner. Regarding this, an adaptation approach supporting individualized gameplay and learner-game interaction has been described and implemented within the affective tutoring system and educational game which is integrated into the system. Furthermore, experimental study with the developed educational game has been done to evaluate its effect on the learning motivation, emotions, and learning results.

978-1-5386-4359-4/18/$31.00 ©2018 IEEE

II. Related Work

A. Serous Games and Competence Development

In an area where competence level and challenge level are well balanced the so-called "flow-experience" comes into effect, see Fig. 1. In the flow model, the occurrence of some emotions like boredom or anxiety shows a mismatch between challenge (difficulty level) and knowledge level, therefore, an arise of such emotions can help to identify, for example, knowledge gaps.

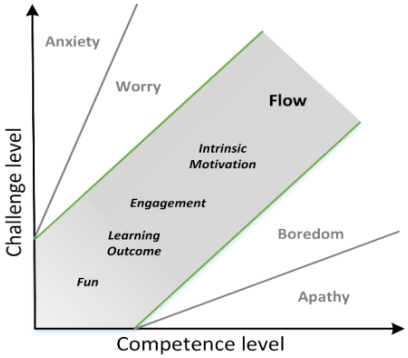


Fig. 1. The Flow-Experience according to Csikszentmihalyi [3].

A good game should keep the user's attention and provide an entertaining user-experience. Moreover, to enhance motivation based on the flow-experience it is of particular importance to find the individual middle course between a challenge and a feeling of success. Exactly this comfort level could be used to promote self-regulated learning processes. Self-regulated learning is understood as a learning process in the sense of a cyclical and reflexive process in which the action steps are directed and managed in an individual goal-oriented manner [11, 12]. By combining positive and negative feedback, challenges and interactivity level it is possible to acquire higher player's satisfaction and keep him/her in the flow state which is considered to be optimal for learning [13].

B. How Serious Games Affect Learning and Motivation?

Back in 2007, Blant [14] explored the effectiveness of game-based learning in the context of higher education in three studies. The students used games, which are essentially simulation games depicting economic correlations in micro- and macro-economic terms. Results showed that students, who were using mentioned games, achieved better overall results in the exams.

For instance, the meta-analysis of Sitzmann [15] provides a possible explanation, in which game-based approaches were used in a classroom and compared against a control group. The study shows positive effects on the memory performance of the test persons, which was shown in a better retention of knowledge. Results also suggest that there is a higher self-efficacy and an increase in declarative and procedural knowledge compared to the considered control group. In support of this, Kato *et al.* [16] have shown a positive influence on the self-efficacy concept in their study. In 1997, Wolfe [17] already found a significant effect on the use of games at the knowledge level as well as on a larger increase in knowledge in the operational context.

In the educational context, a meta-analysis of 67 studies by Randel *et al.* [18] in the area of mathematical competencies also found a significant effect on performance. These findings did not apply to humanities. Randel *et al.* [18] argue that the use of computer games is particularly suitable for clear goals and content. For the subject of mathematics, Ke [19] also researched the effect of strategy games to support problem-solving and decision-making competence. Furthermore, using a study Ke [19] showed an increase in motivation through serious games compared to classical forms of learning. The study of Yang [20] also found positive effects on the motivation of pupils to learn in fields of politics and social sciences as well as an improvement in problem-solving skills.

Hays [21] concluded from 105 studies that for the effect of game-based learning it is fundamental that games are well designed and must pursue specific learning objectives. Unlike Randel *et al.* [18], Hays [21] does not argue out of the perspective to the subject matter, but from a didactic and conceptional point of view of the game with a focus on its learning goals. This means that the didactic goals have to be initially defined in the development process and based on goals the game design has to be embedded in a learning environment. Moreover, it is considered that playing games is an emotional process rather than a rational one [22]. Emotions are an integral part of the learning process since they influence perception, attention, decision making, acquisition and retrieval of knowledge, as well as motivation to learn [23, 24]. During gameplay, various emotional states can arise [13]. If knowledge assessment is carried out by using an educational game, then negative emotions (e.g., anxiety or even fear) can appear as well [25, 26, 27]. Research shows that players are not spending long hours playing a game just to entertain themselves since most of the games can be very frustrating [28]. Therefore, it is considered that people play games to satisfy some basic needs, e.g., a need to learn, need to be challenged, or need to win. In particular, Petko [8] addresses the aspect of embedding learning tasks in the game context and shows in its study that people try to reduce the absorption of knowledge in an explicit learning mode in order to switch back to the game mode as quickly as possible.

Sitzmann [15] also notes that the success of a game is primarily due to the fact that participants had a repeated use of the teaching content and that, in comparison to classical methods, the subject matter was acquired actively and independently. While Wouters *et al.* [29, 30] can confirm the results of Sitzmann [15] regarding knowledge retention, no superiority of active learning in serious games could be found. Results with regard to motivational aspects of games are also inconsistent with the assumption that there is a fundamentally positive relationship between gaming and motivation to learn. Instead, a differentiated examination of the topic is necessary, which requires pedagogical concepts in particular. Their findings lead to the question, whether "we can design serious games in such a way that players are automatically prompted to reflect on their performance during gameplay?" [30]. According to this question, computational intelligence could be a pioneer answer, if the technology will be implemented in a didactic and meaningful manner.

Summarizing, to promote the flow-experience with an aim to support self-regulated learning the learning tasks and processes have to be inherent part of the game design and story on the one hand. On the other hand, the task`s difficulty, instructions and feedback need to meet the individual learner`s condition and

situated emotional state. These findings lead to the assumption that computational intelligence could be the key factor for future educational game development. This would allow supporting individual learning paths and encouraging self-regulated learning processes by reacting on learner`s emotions automatically and providing situated and need-based feedback to keep the learners up with the flow. Furthermore, analysis of existing research regarding educational games, emotions and gameplay individualization allows to conclude that combination of these three aspects have been neglected [31, 32, 33]. Therefore, this research supplements existing studies in this direction both from a theoretical and practical perspective.

III. Computational Intelligence to Support Learning in Games

The web-based affective tutoring system called ELIA (**E**motions for **L**earning and **I**ntelligent **A**ssessment) is developed at the RTU integrating emotion recognition based on facial expressions for the tutoring adaptation purposes. In the system, students can access the personal information and see learning and assessment results in the particular study course [34].

The system is using an emotion-based adaptation method running on two levels: (1) macro-level and (2) micro-level. Adaptation at the macro-level is implemented prior to learning on the basis of static learner's data available to the system (based on personality model, learning style, etc.). Adaptation at the micro-level is ensured during the learning/teaching process based on dynamic learner's data acquired in real time during the learning (based on learner's actions, responses, results, emotions, etc.). This method is implemented in the educational game to support game-based assessment (GBA) [35].

The game integrated as part of ELIA is currently used for teaching topics in the RTU study course "*Fundamentals of Artificial Intelligence*" and assessing knowledge in this course (see Fig. 2). However, it is possible to assess knowledge in other topics or study courses since the game is not limited by course topics and questions from other fields can be added. Knowledge assessment is implemented as a quiz following one of the most popular games called "*Who Wants to Be a Millionaire?*" in which players try to win $1000000 by answering series of multiple-choice questions with increasing difficulty.

Although traditional assessments (e.g., multiple-choice questions) are considered non-motivating and not engaging for learners [13], as well as negatively influencing learning outcomes, recent research suggests that modifications of these traditional assessments with more engaging features (e.g., involving gamification) is a promising approach to GBA [36]. The main aim is to assess learners' knowledge through adapted game elements leading to increased motivation to achieve higher results not only in the game but also in final exams. In the last few years, the failure rate in the exam of the previously mentioned study course has increased notably up to 60%.

To provide adaptation at the macro-level, learner's personality represented as Big Five personality traits is analyzed. The personality traits give also information about learner's default mood or tendency to some specific emotions, preferred learning style, teacher's type, and teaching approach, as well as they allow identifying tendency towards goal achievements [34]. Since this information represents static learner's data then it is used as a basis for macro-level adaptation to select most appropriate teacher's type (friend, expert, coach, or evaluator) represented as pedagogical agent and teaching strategy (based on learner's learning style) and achievement goal – either mastery or performance both in terms of approach (acquisition) and avoidance [37]. This influences the system's (teacher's) behavior and interaction with a learner during the learning and assessment process. To identify these parameters automatically, the personality model has been integrated into the system consisting of two neural network models. Both models were trained based on survey data (personality dimensions, learning style and achievement goal) to allow assigning learning style and achievement goal for learners based on their personality.

Emotion detection is implemented by adopting the existing web-based solution *Emotion API* developed by *Affectiva* [38]. This company has collected more than 6 million face videos in 87 different countries including Central and Eastern European countries allowing recognition of emotions across different nations despite people's age, ethnicity, and gender [39]. This tool allows tracking movement of facial actions, engagement, attention, and recognizes basic emotions based on the analysis of facial expressions acquired from camera stream with 90% accuracy [39]. Since learning specific emotions (e.g., flow, frustration, confusion, boredom and anxiety) are recognized using this tool, studies regarding emotion identification were analyzed to identify typical facial actions for these emotions [40, 41]. The neural network model is used for the classification of detected facial actions to identify mentioned emotions.

Adaptation at the micro-level is implemented on the basis of dynamic parameters and emotions since they are occurring and changing during the gameplay and they can serve as an evidence of flow-experience. Each type of teacher (pedagogical agent) has its own reacting rules developed according to learner's emotional reactions. A detailed description of the adaptation approach both at macro-level and micro-level is provided in [34].

Before starting the knowledge assessment, learners are introduced with playing rules, available additional options and an individual goal which is set for each learner based on identified achievement goal [37]. The main aim of a set goal is to add an extra challenge to the gameplay, for example, '*answer at least to 10 questions without using additional options*' or '*get in the 1st place by answering to all 15 questions in less than 00:15:35*'. During the gameplay, learners need to go through 15 multiple-choice questions of increasing difficulty (5 easy, 5 medium, and 5 hard questions). In addition, after each attempt, the system analyses provided answers (correctness, answering time, emotional reactions during answering) to select for the particular learner in the next attempt previously unseen questions and/or identified problematic questions. Problematic questions can appear in case of incorrect answers, used options and long answering time, as well as based on the analysis of emotional states registered during the question. According to the analysis of interaction, emotions and gameplay the system gives immediate feedback during gameplay and after finishing the game (either winning or losing).

In the next section, an experiment is described involving usage of the developed game for the knowledge assessment during the lectures of the study course and after lectures in students' free time.

Fig. 2. Game interface for knowledge assessment.

IV. Empirical Analysis

The current study investigates whether differences exist between the paper-based assessment and the GBA in terms of learners' performance (gained points) and whether GBA can have a beneficial effect on learners' emotional experiences and motivation and therefore improve learners' performance. Based on this, two research questions were defined:

1) Whether type of knowledge assessment (paper-based or game-based) influences assessment results?
2) Whether engagement in the GBA affects flow-experience and motivation and how this influences learning results?

The methodology followed to accomplish the goal of this study is being described in the following subsections of the paper.

A. Methodology

1) Participants

Participants were 244 (200 male and 44 female) third year RTU undergraduate students who were taking study course "*Fundamentals of Artificial Intelligence*". The participants' average age was approximately 22 and 84% of participants were from Latvia and 16% of participants were foreign students coming from such countries as Germany, France, Uzbekistan, Kazakhstan, India, Turkey, China, Korea and others. Therefore, the experiments were run in both languages – Latvian and English. Since the experiment was organized in two parts – *knowledge assessment* on the one of topics into controlled conditions and *knowledge self-assessment* on two other topics (during the free time) then difference exists between the participants' assignment to the experiment. In the first part, all students attending the study course were randomly assigned to paper-based (N=153) or game-based knowledge assessment (N=87) where assessment results were taken into consideration for the admission to the final exam and calculation of the final grade. In turn, the participation in the second part of the experiment (using only game for knowledge self-assessment) was on a voluntary basis. Compensation in the form of maximum 10% of the final grade was granted for the participants. In the second part of the experiment, 84 participants engaged including 4 students who did not participate in the first study.

2) Instruments

At the beginning of the study course, participants completed paper-based questionnaires, e.g., by adopting widely recognized NEO Five-Factor Inventory (NEO-FFI) to acquire Big Five personality traits representing student's personality. Moreover, questionnaires were used to identify student's learning style according to Kolb's learning styles [42] and 2x2 Achievement Goal Questionnaire [37] to assess students' achievement goals allowing identifying differences between students in terms of motivation to achievements and success.

Before running the first part of the experiment (before knowledge assessment) students were given a questionnaire to answer questions regarding the preferences of assessment type (paper or computer), challenges and benefits experienced when taking a paper-based assessment and when – a computer-based assessment, as well as typical emotions experienced before and during the assessment. A part of questions were represented as semantic differential scale (e.g., 1 indicating "strongly prefer paper-based assessment" and 7 – "strongly prefer computer-based assessment") but mostly open-ended questions were used (e.g., regarding challenges and benefits of assessment types and emotions experienced).

After the knowledge assessment, participants were asked to complete one more questionnaire with open-ended questions regarding their affective experiences during the knowledge assessment and possible emotional differences when taking assessment in the other form, i.e., those who were writing the paper-based assessment were asked about possible emotions during a GBA.

Other parameters, e.g., flow experience, motivation to learn and achieve higher results were estimated based on data collected using the game during the whole experiment, e.g., collected points, attempts done, number of times the student won the game, emotional data, etc.

3) Experimental Design

As mentioned before, the experiment was organized in two parts – at the university during the knowledge assessment activity in the study course "*Fundamentals of Artificial Intelligence*" (compulsory activity) and after classes in students' free time till the final exam (voluntarily activity).

In the first part, each students group (4 groups in total) was randomly divided into two groups – students writing assessment in the class (paper-based assessment) and students working in the RTU computer class using the developed game. Because of limited number of available computers (25 computers) maximum 25 students from each group (depending on group's size) were randomly chosen for working in the computer class. Assignment to one of assessment types is represented in the Fig. 3. In total, 153 students took part in the paper-based assessment and 87 students – in game-based assessment. The study compared the effectiveness of two different assessment methods and evaluated the influence of each assessment type on student's affective experiences. The two methods were identical in terms of assessed knowledge (all students were taught during lectures with the same learning content on the topic "*Uninformed Search*" and were informed about assessment during the next class), both were based on multiple choice tests with 15 questions and all students were provided with three additional options (cross out two incorrect answers (50:50), add "correctness" percentage to answers (ask the audience) and provide explanation to one of student's chosen answers (phone a friend)). All students needed to collect at least 1.25 points (from 2.5 points) to pass the test.

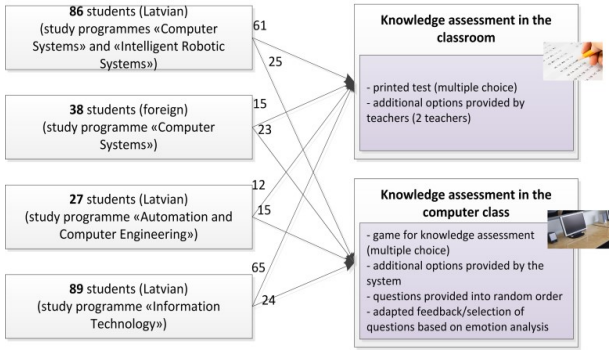


Fig. 3. Assignment of students to experimental conditions.

The main difference was that one of the experimental conditions was in written form with no adaptations provided, while the second one took place in the computer class by playing the game. In the second case, students needed to answer 15 sequenced questions based on learner's knowledge and difficulty level with an aim, first, to win $1000000 and, secondly, to follow the system's set goal depending on the identified achievement goal for each student. The questions on each attempt were adapted to student's knowledge and emotions, immediate feedback on achieved results was provided and automatic system's support was offered to a student based on his/her characteristics, e.g., learning style or recognized emotions, and actions during the gameplay.

Before starting the experiment, all students were gathered in the classroom where they were asked to fill in the questionnaire described in previous section regarding differences in assessment types and students' preferences, as well as emotional experiences during assessments. After the completion of the questionnaire, students who were selected randomly for the game-based assessment were taken to the computer class.

Students working in the class received a printed version of the multiple-choice test with 15 questions and four possible answers. Furthermore, students were informed about additional options they can use during the assessment by reducing number of points they could get for the particular question depending on the difficulty of the question (5 easy, 5 medium, 5 hard questions).

During the knowledge assessment in the computer class, students were given instructions for logging in the web-based system and starting the game, as well as about possibility to see results of the knowledge assessment and summary of completed questionnaires at the first weeks of the study course. They were asked to play at least 5 times (not mandatory, more for data collection purposes) and grant access to the game to connect the camera for the emotion detection. Furthermore, students who didn't want to participate in emotion recognition activity (by working with turned on video camera), were allowed to disconnect camera and play the game without the analysis of their emotional data (3 participants refused to work with turned on camera). If game was started for the first time all instructions regarding the gameplay were provided to the student by the pedagogical agent acting as a tutor in the game. During the experiment, students did not have any questions regarding the game itself or its playing rules since most of them were familiar with the original game "*Who wants to be a millionaire?*". Under both experimental conditions students were not limited with the test completion time (they could use all 90 minutes allocated for the lecture).

After the knowledge assessment in the both experimental conditions students were asked to complete questionnaire with two questions regarding their affective experiences during the knowledge assessment. One question was about their emotions during the experiment and in case of negative emotions students were asked to express their opinion related to the possible differences in emotional experiences if the assessment would have been offered either as computer-based assessment through educational game (to students taking written test) or typical paper-based assessment (to students working in computer class).

Therefore, by changing assessment type and emotion involvement was measured changes in the assessment results and emotional experiences in both experimental conditions. In addition, various other variables were collected during the gameplay, e.g., number of attempts and number of answered questions in each attempt, time needed to answer each question, overall playing time per attempt, set goals and their achievement results, and used options.

In the second experiment part, all students were informed about possibilities to assess their knowledge in two other topics "*Heuristic Search and Game Algorithms*" and "*Knowledge representation*" of the same study course. Instructions for logging in the system and using the game were published in the RTU MOODLE system. The participation was on a voluntary basis after classes in students' free time till the final exam.

To motivate students in the participation they were granted with the maximum 10% of the final grade depending on the assessment results (5 points were given for each of the topic). The main aim of this study was to provide students with more possibilities to practice on topics and prepare better for final exam in this study course. In total, 84 students participated in the self-assessment activity varying from one topic to another. Although, it was not a requirement to carry out assessment on the first topic (evaluated in the first part of experiment)

58 students participated also in this activity. 83 students took part in the self-assessment on topic 2 and 71 students on topic 3.

In this study, the collection of the data was carried out regarding various parameters of a gameplay, e.g., emotional data (if students had cameras connected to computers and used them), number of attempts, given correct/incorrect answers, score, time needed to answer questions, playing time, achievement of set goals and used options, as well as activities were logged (e.g., selected options in the game menu, clicked buttons, etc.) so that it was possible to synchronize emotional changes with student's actions during the gameplay and to analyze action sequences with an aim to identify typical behavioral patterns.

4) *Data Analysis*

Descriptive statistics (means, medians, and standard deviations), and intercorrelations of variables in the experiment were determined using the IBM SPSS Version 23 and the level of significance was set to $p < 0.05$. The analysis of the collected data (assessment results) revealed that the data did not follow normal distribution. Therefore, non-parametric test Mann-Whitney U-test was conducted to compare differences between the results of the two conditions (paper-based and game-based) and Kruskal-Wallis Test for independent samples to evaluate differences between scores among various emotional states. Results of the knowledge assessment were used as dependent variable, while the assessment type or emotional state was considered as the independent variable.

V. Results and Discussion

First, we investigated, whether differences exist between results of the paper-based and game-based assessment in terms of the students' performance represented as gained points in the experiment`s first part of, as well as emotional data collected during the whole experiment was analyzed in terms of student's performance and motivation.

A. Performance Differences

In general, overall statistics of knowledge assessment results in both assessment types is the following:

- in paper-based assessment, 92.8% (N=142) of students passed the test (got ≥1.25 points) and 7.2% of students (N=11) failed in the test;
- in game-based assessment, 82.8% of students (N=72) passed the test, in turn, 17.2% (N=15) of students failed.

Considering the fact that most of the students passed the test in both conditions it can be concluded that offered test was quite easy for most of them, however, it must be noted that everyone was informed about the assessment in the previous lecture.

To compare differences between results (gained points) of both assessment groups, a non-parametric Mann-Whitney U-test was selected. After running the test, significant differences were found between both groups. The test indicated that assessment results were higher in the paper-based test (*Mdn= 1.95*) than in the game-based test (*Mdn= 1.65*, *U= 5459 (Z= –2.32), p= .020*). However, further analysis of the differences showed small effect size between both groups (*r= –.149*). Estimated results indicate that differences in the assessment type have small effects on assessment results. Furthermore, direction of the effect shows small existence of negative correlation between assessment type and students' results. Therefore, it can be concluded that type of the knowledge assessment itself is not the determining factor affecting scores. These results can also be explained by done analysis of the question answers regarding preferences of assessment type (paper-based or computer-based) given by students before the experiment. Results showed that overall there is no preference regarding assessment type (*M= 4.10, SD= 1.88*, where 4 indicated 'no preference').

Such differences in the assessment results can be explained by various reasons. Firstly, despite the fact that students writing the paper-based test were required to sit alone and two teachers controlled the process still possibility existed that students could find ways how to get correct answers and/or compare them with each other since order of the questions for paper-based test was the same for all students. Secondly, the same test was used for each group, therefore, might be a situation that offered questions were communicated to other students taking the test later. In turn, lower results in GBA can be explained by the fact that sequence of the questions differed, and students were more motivated to concentrate on their own work and not rely on others. All these aspects might have potentially affected the assessment results and differences between both conditions.

B. Flow experience, Motivation and Influence on Learning

Indicators for the facilitation of motivation can be represented as spent time on playing, number of attempts done in the game, as well as reaching of 15th question to win the game. In the first part of the experiment, after comparing times spent on the assessment in the classroom and computer class, quite significant differences were observed. Most of the students who were writing the paper-based test (with 15 questions) completed it in approximately 20-25 minutes. In turn, students working in the computer class needed less time to complete (or win) the game (average time: 00:06:41) by answering to all 15 questions. The fastest victory was in 2 minutes (longest playing time: 22 minutes). Overall, even though students who were playing the game were asked to do at least 5 attempts for experimental purposes (data collection), they continued playing till the end of the lecture time and even longer (maximum 27 attempts were done by one of students). In average, students made ~10 attempts (*M= 9.91, SD= 5.96*) what is more than it was asked at the beginning of the experiment. Therefore, involvement in the gameplay and willingness to get higher scores can be a good indicator of increased motivation and student's engagement leading to better learning results. On other two topics the number of attempts were respectively (*M= 5.55, SD= 4.71*, maximum 25 attempts) and (*M= 6.91, SD= 7.12*, maximum 40 attempts).

One benefit of GBA relies on positive affective outcomes facilitating learning outcomes. We studied the affective outcomes with respect to flow experience and the main question to answer was whether students experiencing positive emotions receive higher learning results? To answer to this question, emotional data and other parameters (e.g., points, playing attempts, etc.) from the system were analyzed. During the whole experiment, 615391 emotional states were registered using the system. Total number of emotional states made during the gameplay was as follow: flow (236288; 38%), anxious (144422; 23%), bored (92662; 15%), neutral (54492; 9%), surprised (50811; 8%), frustrated (13324; 2%), happy (11851; 2%), confused (35; 2%), and sad (7535; 1%). The most common registered emotional states (flow, anxiety and boredom) can also

serve as an evidence of flow-experience since movement through these states is a result of continuous interaction between task difficulty (challenge) and progression of learner's skills.

Differences between assessment results on each topic in terms of emotional data (positive or negative) were considered. Who scored higher – those students who had positive or negative emotions? To answer this question descriptive statistics (see Table 1) regarding scores were acquired for each topic by emotional categories – 'no emotional data' if the student was not playing the game with emotion analysis, 'more negative' if the student experienced more negative emotions during the gameplay, and 'more positive' if positive emotions were more often registered by the system during the playing process.

TABLE I. AVERAGE SCORES BASED ON EMOTIONAL DATA

Results on Topics		*N*	**Descriptive statistics**	
			Mean	
			Statistic	***Std. Deviation***
Results on Topic 1	no emotional data	42	1.70	0.59
	more negative	44	1.71	0.57
	more positive	41	1.73	0.57
Results on Topic 2	no emotional data	64	3.85	1.23
	more negative	6	3.92	0.52
	more positive	13	3.68	1.57
Results on Topic 3	no emotional data	54	3.16	1.42
	more negative	8	4.01	0.89
	more positive	9	3.32	1.39

More equal sample size for each category was acquired on topic 1 since many students were involved in first part of the experiment and played the game using the emotion recognition functionality. Results of the analysis show that there is slightly higher scores for students who experienced positive emotions (*M= 1.73, SD= 0.57*) from maximum 2.5 points. In turn, students who were using the system without the emotional analysis performed a bit worse (*M= 1.70, SD= 0.59*). However, non-parametric Kruskal-Wallis Test for independent samples didn't show significant differences ($p>.05$).

Statistics on the other two topics, where students could get maximum 5 points, is quite poor regarding emotional data – 19 and 17 students used the system with emotional analysis. Other students played the game without emotion involvement (for example, because of missing camera or just not wishing to turn it on). Overall, in both cases better results showed students who experienced more negative emotions than positive or played without analysis of emotional data. Also, in these two cases significant differences were not found ($p>.05$). However, more detailed analysis of gained scores for the last two topics allows concluding that topic itself has an influence on the results. If compared gained points on topic 2 and topic 3 (from max 5 points) then in overall higher scores were acquired in topic 2 (*M= 4.02, SD= 1.06*) compared to topic 3 (*M= 3.30, SD= 1.39*). This can also be explained by pedagogical experience in the particular study course that shows that the last topic has been the most complicated for students and similar decrease in scores can be observed in exams regarding these two topics for many years.

VI. CONCLUSIONS

A comprehensive literature review has been done in terms of serious games and their influence on learning, motivation, emotions, particularly on flow-experience. Computational intelligence has been discussed as a key factor for future serious games aiming to support individual learning paths and promote learner-game interaction. The developed educational game which uses emotion-based adaptation is described. Emotions are considered as one of the parameters for the tutoring adaptation since they are occurring and changing during the gameplay and can serve as an evidence of flow-experience.

The experimental study was conducted with 244 participants involving usage of the developed game for the knowledge assessment during the lectures of the study course and after lectures in learners' free time. Results of the experiment`s first part showed that learners using the game got lower scores, however, it was concluded that changes in assessment type (paper or game-based) have small effect on assessment results and it is not the determining factor affecting scores. Furthermore, usage of the game can be beneficial for low-performing students since some of them in the questionnaire given after the experiment mentioned that by using the game their knowledge level increased after the assessment.

Further data analysis of the involvement in the gameplay indicated on increased motivation and student's engagement in the GBA. Usage of the game facilitated motivation since most of the students did more than it was required and at homes playing attempts were also higher than required. From the pedagogical and psychological point of view, it can be said that further research is needed. On the one hand, to show exactly how serious game design and the combination of specific design elements affect learning processes and motivation. On the other hand, further research is needed to point out how learner-game interaction can influence self-regulated learning and support self-reflection processes. . Only then game potential can really be used to design learning environments with a specific didactic value to promote self-regulated learning activities.

Analysis of emotional data collected during the experiment, showed that slightly higher scores got students who experienced positive emotions. Therefore, an important step in the future work would be the collection of more data (particularly, emotional data) to find stronger evidence for supporting relationship between positive emotions and higher performance.

Furthermore, observations show that fear still exists from new technologies and not all students are open to the analysis of their emotional data and usage of cameras. Therefore, possible other methods for the emotion identification which do not influence learner's attitude negatively towards the game should be considered for the development of not only educational games but other learning environments. Regarding this issue, a possible solution can be an integration of emotion recognition approaches which are not based on the classification of sensor data, e.g., data from camera. However, additional research is required in this direction since current approaches do not provide sufficient accuracy of emotion recognition [43] and thus can crucially decrease accuracy of the system's behavior adaptation.

Moreover, it is necessary to extend the target group to non-academic learners and people with little experience in using specific learning strategies and self-regulated learning. Attention will be drawn here, especially, to support this kind of learners by using serious games.

REFERENCES

[1] S. Deterding, D. Dixon, R. Khaled, and L. Nacke, "From game design elements to gamefulness: Defining "gamification"," in Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments, 2011, pp. 9-15.

[2] S. Deterding, "Gamification: Design for motivation," Interactions, vol. 19(4), 2012, pp. 14-17.

[3] S. Kaczmarek, N. Straub, and M. Henke, "How serious games unfold their potential in further training in logistics. Results of a multiperspective rmpirical requirements analysis," in Proceedings of the 9th annual International Conference on Education and New Learning Technologies, 2017, pp. 9583-9591.

[4] R. Ghali, S. Ouellet, and C. Frasson, "LewiSpace: An exploratory study with a machine learning model in an educational game," Journal of Education and Training Studies, vol. 4(1), 2016, pp. 192-201.

[5] C. Meier and S. Seufert, "Game-based learning: Erfahrungen mit und perspektiven für digitale lernspiele in der beruflichen bildung," in Handbuch E-Learning: Fachverlag Deutscher Wirtschaftsdienst. Köln, 2003, pp-1-17.

[6] C. DeWitt and S. Ganguin, "Kommunikation in serious games," in Digitale Lernwelt – Serious games: Einsatz in der beruflichen Weiterbildung. Bielefeld: Bertelsmann-Verlag, 2011, pp. 97-108.

[7] J. Hamari, J. Koivisto, and H. Sarsa, "Does gamification work? – A literature review of empirical studies on gamification," in Proceedings of the 47th Annual Hawaii International Conference on System Sciences, 2014, pp. 1530-1605.

[8] D. Petko, "Unterrichten mit computerspielen. Didaktische potenziale und ansätze für den gezielten einsatz in schule und ausbildung," MedienPädagiogik: Zeitschrift für Theorie und Praxis der Medienbildung, vol.15, 2008, pp.1-15.

[9] S. Deterding, "Serious games. Game-based-learning. Chancen und grenzen," Educamp Ilmenau, TU Ilmenau 19.-20.02.2008.

[10] S. Kaczmarek, N. Straub, and M. Henke, "How to promote self-regulated-learning processes by using serious games," in Proceeding of 10th annual International Conference of Education, Research and Innovation, 2017, pp 3670-3679.

[11] D. Euler, M. Lang, and G. Pätzold, Selbstgesteuertes Lernen in der Beruflichen Bildung. Steiner Franz Verlag, 2006.

[12] K. Jonas, W. Stroebe, and M. Hewstone, Sozialpsychologie. 6. Vollständig überarbeitete Auflage. Berlin:Springer, 2014.

[13] J.L.Sabourin and J.C. Lester, "Affect and engagement in game-based learning environments," IEEE Transactions on Affective Computing, vol. 5(1), 2014, pp. 45-56.

[14] R. Blunt, "Does game-based learning work? Results from three recent studies," in Proceedings of the Interservice/Industry Training, Simulation, & Education Conference, 2007, pp. 945-955.

[15] T. Sitzmann, "A meta-analytic examination of the instructional effectiveness of computer-based simulation game," Personnel Psychology, vol. 64(2), 2011, pp. 489-528.

[16] P.M. Kato, S.W. Cole, A.S. Bradlyn, and B.H. Pollock, "A video game improves behavioral outcomes in adolescents and young adults with cancer: A randomized trial," Pediatrics, vol. 122(2), 2008, pp. 305-317.

[17] J. Wolfe, "The effectiveness of business games in strategic management work," Simulation & Gaming, vol. 28(4), 1997, pp. 360-376.

[18] J.M. Randel, B.A. Morris, C.D. Wetzel, and B.V. Whitehill, "The effectiveness of games for educational purposes: A review of recent research," Simulation & Gaming, vol. 23(3), 1992, pp. 261-276.

[19] F. Ke, "Computer games application within alternative classroom goal structures: Cognitive, metacognitive, and affective evaluation," Educational Technology Research and Development, vol. 56(5-6), 2008, pp. 539–556.

[20] Y.-T.C. Yang, "Building virtual cities, inspiring intelligent citizens: digital games for developing students' problem solving and learning motivation," Computers & Education, vol. 59(2), 2012, pp. 365–377.

[21] R. T. Hays, The Effectiveness of Instructional Games: A Literature Review and Discussion. Orlando, FL: Technical Report, Naval Air Warfare CenterTraining Systems Division, 2005.

[22] D. Bonnycastle, "Promoting active learning using games," Active Learning Series, 2009. [Online]. [Accessed: Mar. 9, 2017].

[23] N. Schwarz, "Emotion, cognition, and decision making," Journal of Cognition and Emotion, vol. 14(4), 2000, pp. 440–443.

[24] B. Lehman, S.K. D'Mello, and N. Person, "The intricate dance between cognition and emotion during expert tutoring," in Proceedings of 10th International Conference on ITS (Part II), 2010, pp. 433-442.

[25] A. Mavridis and T. Tsiatsos, "Game-based assessment: Investigating the impact on test anxiety and exam performance," Journal of Computer Assisted Learning, vol. 33(2), 2017, pp. 137-150.

[26] S.K. D'Mello and A.C. Graesser, "Feeling, thinking, and computing with affect-aware learning technologies," in The Oxford Handbook of Affective Computing, 2015, pp. 419-434.

[27] M. Zeidner, "Test anxiety in educational contexts: Concepts, findings, and future directions," in Emotion in Education, 2007, pp. 165–184.

[28] K. Becker, "Pedagogy in commercial video games," Games and Simulations in Online Learning. Research and Development Frameworks, 2007, pp. 21-47.

[29] P. Wouters, E.D. Van der Spek, and H. Van Oostendorp, "Current practices in serious game research: A review from a learning outcomes perspective," in Games-Based Learning Advancements for Multi-Sensory Human Computer Interfaces: Techniques and Effective Practices, 2009, pp. 232-250.

[30] P. Wouters, C. van Nimwegen, H. van Oostendorp, and E.D. van der Spek, "A meta-analysis of the cognitive and motivational effects of serious games," Journal of Educational Psychology, vol. 105(2), 2013, pp. 249-265.

[31] C. Conati and M. Gutica, "Interaction with an edu-game: A detailed analysis of student emotions and judges' perceptions," International Journal of Artificial Intelligence in Education, vol. 26(4), 2016, pp. 975-1010.

[32] J. Li, Y. Han, and S. Liu, "Physiological evaluation of the players' emotions in different educational games," American Journal of Educational Research, vol. 2(9), 2014, pp.735-739.

[33] R. Sawyer, A. Smith, J. Rowe, and R. Azevedo, "Enhancing student models in game-based learning with facial expression recognition," in Proceedings of the 25th Conference on User Modeling, Adaptation, and Personalization, 2017, pp. 192-201.

[34] S. Petrovica and A. Anohina-Naumeca, "The adaptation approach for affective game-based assessment," Applied Computer Systems, vol. 22, 2017, pp.13-20.

[35] D. Ifenthaler, D. Eseryel, and X. Ge, "Assessment for game-based learning," in Assessment in Game-Based Learning. Foundations, Innovations, and Perspectives, 2012, pp. 3-10.

[36] B. Lehman, D. Hebert, T. Jackson, and L. Grace, "Affect and experience: Case studies in games and test-taking," in Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems, 2017, pp. 917-924.

[37] A.J. Elliott and H.A. McGregor, "A 2 × 2 achievement goal framework," Journal of Personality and Social Psychology, vol. 80, 2001, pp. 501–519.

[38] Affectiva, "Products – SDK&API," Affectiva, 2017 [Online].

[39] Affectiva, "Emotion AI 101: All About Emotion Detection and Affectiva's Emotion Metrics," 2017 [Online].

[40] S.K. D'Mello and A.C. Graesser, "Confusion," in International Handbook of Emotions in Education, 2014, pp. 289–310.

[41] J.F. Grafsgaard, J.B. Wiggins, K.E. Boyer, E.N. Wiebe, and J.C. Lester, "Automatically recognizing facial indicators of frustration: A learning-centric analysis," in Proceedings of International Conference on Affective Computing and Intelligent Interaction, 2013, pp. 159–165.

[42] L. Mackevica, "Teaching aid "Learning Styles"," in "Topicalities in Pedagogy for Professional School Teachers", 2010 [Online].

[43] A.F. Botelho, R.S. Baker, and N.T. Heffernan, "Improving sensor-free affect detection using deep learning," in Proceedings of the 18th International Conference on Artificial Intelligence in Education, 2017, pp. 40-51.

A Virtual Agent Toolkit for Serious Games Developers

Samuel Mascarenhas, Manuel Guimarães, Rui Prada, João Dias and Pedro A. Santos
INESC-ID and Instituto Superior Técnico
Universidade de Lisboa
2744-016 Porto Salvo, Portugal
samuel.mascarenhas@gaips.inesc-id.pt,
manuel.m.guimaraes@ist.utl.pt
rui.prada@tecnico.ulisboa.pt,
pedro.santos@tecnico.ulisboa.pt,
joao.dias@tecnico.ulisboa.pt

Kam Star, Ben Hirsh and Ellis Spice
PlayGen
8-9 Talbot Court, London, UK
kam@playgen.com,
ben@playgen.com,
ellis@playgen.com

Rob Kommeren
Stichting Praktijkleren
3821 AR Amersfoort, NL
r.kommeren@stichtingpraktijkleren.nl

***Abstract*—The design of serious games requires developers to tackle pedagogical challenges calling for advanced solutions that the entertainment industry might deem too risky to pursue. One such challenge is the creation of autonomous socially intelligent characters with whom players can practice different social skills. Although there are several architectures in the field of virtual agents that are designed specifically to enable more human-like interactions, they are still not widely adopted by game studios that develop serious games, in particular for learning. In this paper, we present a virtual agent toolkit that was specifically developed with the intent of making agent-based solutions more accessible and reliable to game developers. To this end, a collaborative effort was established with a game studio that has used the toolkit to develop two different serious games. Among other advantages, the toolkit facilitated the inclusion of a dynamic model of emotions that affects not just how the character looks and acts but also how the player's performance is determined.**

***Index Terms*—serious games, virtual agents, authoring tools, interactive storytelling, affective computing**

I. Introduction

The industry of video games has seen tremendous growth to the point that the budget for highly anticipated games can surpass the cost of big Hollywood films [7]. This led to extensive development times and quite large development teams and corresponding high expectations from the players [13]. On one hand, this state of affairs has enabled the creation of very detailed game worlds with stories and characters that players find very engaging to interact with. But, on the other hand, the huge risk that is now associated with failing to meet the expectations of players has led the industry to primarily focus on what has been known to work in the past. This is also then reflected in the available development tools, with popular game engines like Unity[1] being primarily designed to support the typical requirements and methods used in entertainment games that were previously successful. As a result, game developers that are interested in developing games with more unique characteristics or requirements, which is often the case for pedagogical games, usually find themselves having to spend a significant amount of time in developing their own tools and methods.

The serious games industry is growing as well, supported by the continuous research on the potential in using games for other purposes than just entertainment [10], [12], [21]. Serious games can be used to train and teach players on various subjects (e.g. math fractions [14], logic operators [16]) or raise awareness on social issues (e.g. sustainability [17], cultural diversity [4], bullying [20]). In fact, one of the more interesting aspects in developing games that are designed to teach is that their design is centered around pedagogical challenges. As such, even if the game is very engaging for players it will still fail to achieve its purpose if it does not have a pedagogical outcome. But, in turn, the game might have great pedagogical content but fail to deliver it in an engaging manner. One of the important aspects that make players engaged in a game world is the appeal of its characters. Particularly, non-player characters provide the opportunity for the player to engage in social interactions in a safe environment and within the confines of the game rules and structures. From a training perspective, players are free to experiment and observe the effects their actions have on simulated others in order to obtain and practice certain social skills. However, the range of social interactions that are typically offered to players is still quite limited when compared to real human interaction.

With the goal of expanding the range and complexity of social interactions between characters and humans, there has been a substantial amount of research dedicated to the creation and study of virtual agents. These are embodied characters that are designed to be able to interact with humans in a natural manner [8]. The architectures that have been developed for these characters can be rather complex, having to deal with the challenges of interpreting and synthesizing both verbal and non-verbal actions as well as modeling cognitive and affective processes related to decision making.

Although researchers have been able to successfully apply

[1] https://unity3d.com/

978-1-5386-4359-4/18/$31.00 ©2018 IEEE

virtual agent architectures in the development of serious games (e.g. [1], [9], [11]), such architectures have not yet been widely adopted by game studios. While the accessibility of these architectures can be improved through the creation of better graphical user interfaces and more extensive documentation, there are also technical and conceptual issues that must be addressed [18]. A virtual agent architecture relies on a type of authoring that is oriented towards cognitive concepts such as goals and beliefs, which are quite familiar for AI researchers but not necessarily so for game developers. Also, an agent model will promote a type of storytelling experience that is distributed or character-centric [2] whereas popular game developer tools like *Articy:draft*[2] or *Twine*[3] are designed towards a plot-centric approach with branching dialogues. While these tools can be used to create complex narratives they make a strong distinction between the player and the other characters, by giving dialogue options to the former but not the latter. In the proposed toolkit, while certainly possible, it is not necessary to tie dialogue options to a specific character or the player.

In this paper, we present a novel toolkit that aims to promote the adoption by game developers of virtual agent tools for creating game characters that are more socially and emotionally intelligent (e.g. are able to adapt to the situation and to the players). The toolkit is based on the existing FAtiMA Modular architecture [5], which is an architecture that was has been successfully used in the past in several research applications [1], [3], [4]. These improvements were derived from a close collaboration with game developers at the company PlayGen[4] that used the toolkit to develop two games for learning. The first one is named Space Modules Inc and is being developed for an educational institute in the Netherlands named Stichting Praktijkleren[5]. The game is designed to teach its players how to provide better customer service in technical support. The second game is named Sports Team Manager and is being developed for OKKAM[6], a spinoff company of the University of Trento in Italy. It is a single player game where players assume the role of a sailing team manager. Players must hire, fire and communicate with their team members in order to succeed and, therefore, learn some personel managemnet skills.

This collaboration is part of the ongoing RAGE project[7], which is an EU-funded project with the goal of developing and promoting new technologies for directly supporting applied game developers at creating better applied games and in a manner that is more cost-effective [19].

II. FAtiMA Toolkit

FAtiMA Toolkit is an open-source project[8] that contains a collection of tools and libraries with the aim of enabling the creation of interactive storytelling scenarios with non-player characters that can interact socially with human players in a variety of contexts.

Storytelling can bring multiple benefits to serious games [15]. Not only are people more likely to remember what they learned if the content is integrated in the context of a narrative, but also, an emotionally engaging story will greatly motivate players to achieve the intended learning goals of the game. This form of storytelling centers on the ability of players to shape how the story unfolds according to their actions, as participants rather than as observers. This feeling of agency increases player engagement and encourages them to reflect more deeply on the consequences of their choices. However, the more freedom given to players, the more difficult it becomes to use a traditional scripting approach to author the scenarios. This is because the branching factor of possible narrative paths quickly becomes intractable.

Our proposed storytelling framework deals with this issue by following a character-centered approach rather than a plot-centered one. The authoring is thus focused around the different roles that the characters might play in the game and the narrative emerges from how the characters behave in their given roles. The challenge then becomes to author these roles in a way that characters act in a believable manner but also serve the intended learning goals of the scenario.

As previously mentioned, the toolkit is the result of several improvements that were made to the FAtiMA Modular architecture [5]. For example, the code was ported from the Java language to C# in order to streamline the integration with game engines, such as Unity3D. Also, each component within the toolkit is able to fully load and save its internal state to a *JSON* file. As such, it is possible for the game developer to use his or her text editor of choice to do any kind of authoring task. However, the toolkit contains some complex data structures that refer to one another, such as emotions, an autobiographical memory, appraisal rules, among others. For this reason, each component has an authoring tool with a graphical user interface that help users' in the creation of content in a declarative way preventing syntactical errors. The fact that the entire internal state of each component within the toolkit can be written to a file also works as a logging mechanism.

Many agent-based tools are designed to function as a framework or as a stand-alone application that the game must communicate with, using a specific protocol. In both of these cases, the game developer has to accommodate the game to how the agent tool specifies its communication protocol, its execution cycle and its extensions points, instead of the other way around. Moreover, given their opinionated nature, agent-based frameworks are difficult or even impossible to compose together. It was based on these limitations that we applied a functional library design pattern in the development of the toolkit. Consequentially, all the different components were developed as libraries, i.e. a collection of functions with well defined inputs and outputs, that the game developer can directly import and explore more easily without having to

[2]https://www.nevigo.com/en/articydraft
[3]http://twinery.org
[4]http://playgen.com/
[5]https://www.stichtingpraktijkleren.nl
[6]http://www.okkam.it/
[7]http://rageproject.eu
[8]https://github.com/GAIPS-INESC-ID/FAtiMA-Toolkit

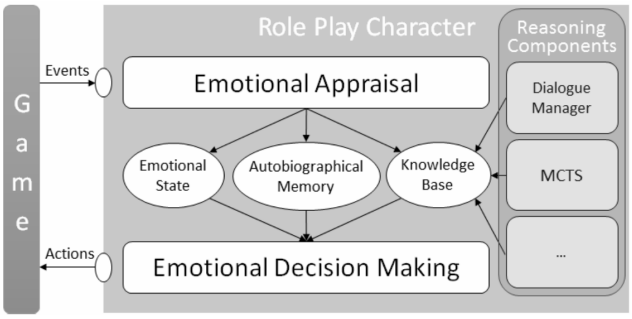


Fig. 1. Diagram of the Role Play Character Component.

worry about future compatibility issues with other tools.

The main functionality of FAtiMA Toolkit is divided in two main components, the Role-Play Character and the Integrated Authoring Tool.

A. Role-Play Character

The Role-Play Character (RPC) is the name given to the component (see Figure 1) within the toolkit that manages each character's reasoning and emotional state based on a perception-action mechanism, which can be described in the following manner. Firstly, the events that occur in the game world are sent as input to the Emotional Appraisal component, which is based on a formalization of the OCC cognitive theory of emotions [6]. This component then determines if the event will trigger a new emotion for the character. Each character can be configured with different appraisal rules that will result in having different emotional outcomes for the same events. After the emotional appraisal process is done, any resulting emotion is added to the Emotional State. Events are also stored in the character's Autobiographical Memory along with any emotion associated to them. The character's Knowledge Base keeps track of what the character believes as logical predicates such as *Weather(Outside) = Raining*. These beliefs are also updated according to the events sent by the game world.

After all the internal structures are updated, the RPC uses the Emotional Decision Making component to select the next action of the character. This is done using a rule-based mechanism that considers both the beliefs of the character as well as its emotional state. In addition to regular beliefs that are directly stored in the Knowledge Base, the decision-making process also takes into account meta-beliefs, which are added by Reasoning Components such as the Dialogue Manager or the MCTS. Syntactically, meta-beliefs are expressed in the same manner as regular ones. The key distinction is that, rather than being stored, the values of these beliefs is determined dynamically by the algorithm specified in the reasoning component. This allows the combination of multiple decision-making strategies into a unified rule-based system. Developers can also register their own modules as additional reasoning components and the meta-beliefs they introduce will become available in the conditional rules of all other components. For instance, consider a game with a specific scoring mechanism for the player and the developer wants to create a decision rule for NPCs to congratulate the player whenever the player's score reaches a certain threshold. This could be achieved by registering the scoring mechanism as a new Reasoning Component that would add *Score(Player) = [x]* as a new meta-belief.

Game characters should have believable emotional responses to give the illusion of life. For applied games that rely heavily on social interaction, it quickly becomes impractical to manually script all the emotional reactions of each character for each possible event. The RPC asset tackles this issue by allowing game developers to create general profiles of how characters respond emotionally in their games. They can test and configure these profiles outside of the game and they can naturally switch between profiles without having to recompile the game source code.

B. Integrated Authoring Tool

The Integrated Authoring Tool is the other main component of the toolkit that is designed to be the central hub for game developers when creating a new storytelling scenario or adapting existing ones. It allows the configuration of the general aspects of the scenario and provides quick access to the authoring tools of the Role-Play Character component. However, the main feature of this component is that it contains a dialogue editor that allows the developer to specify the dialogue acts that are available for both the player and the characters.

For the purpose of dialogue management, the author must define the interaction state where each dialogue may occur as well as define the next state if a certain dialogue is selected. During runtime, all characters are informed about the existing dialogue acts as well as dialogue states. Characters are then able to use this information to decide what to say according to their internal state and decision-making mechanisms. To give an example, consider that the integrated authoring tool informs a character that at the start of the interaction there are two valid dialogues, one to greet the player respectfully, another to greet the player in an angry manner. If the character is angry, the emotional decision making asset will select the second option. If not, then the first greeting will be selected instead.

III. Case Study 1 - Space Modules Inc

Space Modules Inc is a single player game where the player takes on the role of a customer service representative for a spaceship part manufacturer "Space Modules Inc". The virtual characters in the game play the role of customers that call the player (see Figure 2) about hardware and software faults they are experiencing. Some characters will be angry, others uncooperative or stressed, and it's up to the player to manage the situation and decide how best to respond.

Players have to respond to situations by engaging in conversation with customers. This is done by having the player pick one of the available dialogue options in response to the character's chosen dialogue. The process is repeated until the

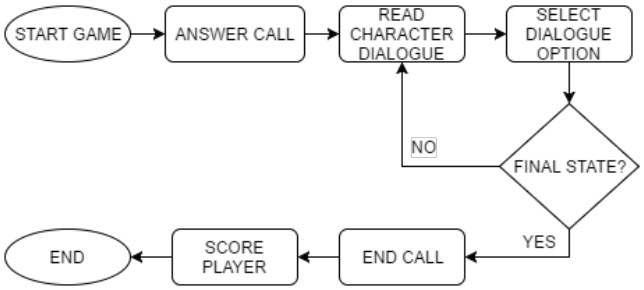


Fig. 2. Space Modules Inc. Game Flow.

Fig. 3. Space Modules Inc - Dialogue Screen (left image) and Result Screen (right image) Flow.

final state of the conversation is reached and then the player's score is passed to the review screen to be shown to the player (see Figure 3). The customer satisfaction score depends on how the player affected the emotional state of the character. The idea is that each customer can have a different emotional profile, thus providing a different challenge to the player. From a pedagogical perspective, players must learn how to manage intense emotions and how to respond to customers in a professional manner in the best way. In other words, the pedagogical goal of the game is to train players in being able to identify a person's emotional state through verbal and nonverbal feedback and gain further experience in providing effective emotional responses.

The emotional reactions of the customers in Space Modules are determined by the Role-Play Character component. According to the selected emotional profile, this component initializes the overall mood of the character to a given value between -10 to 10. The component then updates this value based on how it evaluates the option selected by the player. If the player decides, for instance, to give the wrong solution for the problem that the customer has, the component will generate a "Distress" emotion and the overall mood decreases. The player can then repair the mood of the character by selecting a dialogue that shows empathy for the character's distress.

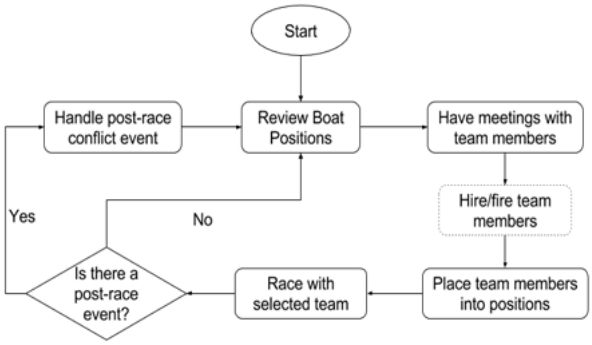


Fig. 4. Sports Team Manager Game Flow.

However, if this dialogue is selected when the character is not feeling distressed, then it will be judged negatively instead and the mood of the character decreases accordingly. The amount by which the mood decreases or increases is also another parameter that is possible to configure in the RPC component.

IV. Case Study 2 - Sports Team Manager

Sports Team Manager is an applied game also developed by PlayGen with the assistance of the FAtiMA Toolkit. The overall goal of the game is to have the player be able to assemble together the most optimally performing sailing team by resolving conflicts and managing the team's interactions. The player interviews virtual characters to identify their skills and personalities. The team has a set of roles, each with overlapping skill requirements. A successful sailing team is not solely based on skill, but also on the social relationships between team members. Players must communicate with their team, deciding which members are placed into each position per race and resolve conflict situations as they arise. Figure 4 shows the game flow during an individual race session.

The players must first review the positions they need to fill on the boat, taking note of the required skills for each. Next, they must meet with their NPC team members, taking into account the skills and inter-team relationships already known, asking questions where further information is needed. Using this information they should, if required, recruit new members into the team and place individuals into positions. After racing with the selected line-up, players will occasionally have to handle events with team members. After the event stage concludes, using the result and pieces of feedback from the race session, players begin the gameplay loop again, but now with additional information to assist in their decision making.

The Role-Play Character component is used here to model the emotional state and decision making of each team member based on their belief set. The component analyses the actions of the player and determines their effect on the emotional state of each NPC based on their current state and the emotional weighting of the event in their perspective. To give an example,

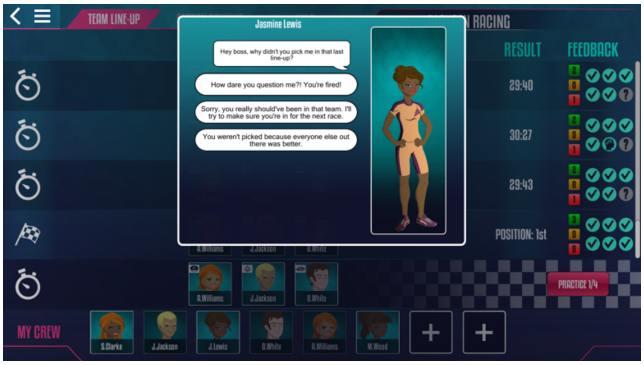


Fig. 5. Sports Team Manager - Post-Race Event.

after each race session, it is possible for a team member to come to the player in order to talk to them. The character might for instance, ask why she was not picked (see Figure 5). Players can then reply back to the team member by selecting from a list of dialogue options. If the player selects an overly aggressive reply, the character is likely to feel angry, affecting its next response.

As mentioned previously, the Role-Play Character component stores the beliefs of every NPC and saves these beliefs over multiple play sessions. These beliefs are related to information such as their last position in the team, skill ratings, opinion ratings and event states. Furthermore, the events sent to the characters are saved, meaning a history of events can be preserved. This allows a history of every team selection to be stored. As all of this information is stored regularly, it can be also be reloaded in further play sessions, allowing for the possibility of a persistent game.

Concerning the Integrated Authoring Tool, this component is used to manage the configuration of the scenario, which contains a list of all possible role-play characters that are dynamically created at the beginning of and during each game. The component also contains all of the dialogue options for the player and the NPCs during various parts of the game, such as team member meetings and post-race events.

V. Game Developers Feedback

Game developers from PlayGen were independent in the integration of the FAtiMA toolkit in their game code and were successfully able to use the toolkit to support the intended gameplay in the two games. They relied on the documentation and examples created for the community and had full access to the toolkit source code. We conducted an informal interview to get their impression regarding the technical integration and the usefulness of the toolkit. Contacts were made by email and face to face. The conversation was around three main questions: (1) How was the FAtiMA toolkit used in the development of the game?, (2) What were the main benefits of using the FAtiMA toolkit? and (3) What were the main difficulties of using the FAtiMA toolkit?

Game developers reported that “the integration was not difficult, but that a proper use of the toolkit requires a steep initial learning curve”. The toolkit facilitated the creation of mechanisms “to determine the change in emotional state and mood depending on the dialogue chosen by the player” and was also useful “to calculate the NPC response to the provided piece of player dialogue, depending on their emotional state and the type of player dialogue selected.” and to “decide how a NPC should greet the player depending on their current relationship with the player.”. They highlighted two main benefits regarding the pedagogical value that the FAtiMA toolkit provided. First, the use of the toolkit was “good because players get immediate implicit (contextual) feedback”. They mean that the emotional responses of the characters were potentially very good cues for the players to assess if they were playing well without the need to show explicit numeric score. The second benefit, was the “ability to dictate the course of conversation indirectly through using the toolkit’s dialogue and NPC emotions systems, as these have made setting up and controlling scenarios a much easier process as a result.”. What is relevant, in the pedagogical sense, is the fact that the definition and setting up of the scenarios was made directly by the trainers who will apply the games. Hence, the game can be configured and adapted by the people who have the most knowledge about the content to be delivered in order to achieve the learning goals of the game.

VI. Students Game AI Projects

The toolkit was also put to test in a course on Game AI at IST, University of Lisbon in the fall semester. It was used in the final project of the course (out of 4) that constituted 30% of the grade. Sixty-eight students, working in groups of three, were engaged. They had a workshop on the FAtiMA Toolkit (of about 2 hours) before tackling the problem. They used a version of the toolkit that is integrated with the Unity game engine and uses components to realise the body and expression of the characters developed by other members of the RAGE project.

Each group was given the task of using the FAtiMA toolkit to create two conversational scenarios, one with a single character interacting with the player and another with two characters engaging in conversation with the player at the same time. Students were free to select any theme for the conversation as long as the non-player characters had believable emotional responses and could be configured to have different personalities. All groups managed to finished the project. Some of the scenarios created had quite interesting and surprising themes. For instance, one group chose to create a scenario where players were at the gates of heaven and had to convince the gatekeeper to let them in. To be successful, players had to avoid upsetting the gatekeeper too much. Other groups opted for a more serious theme such as a job interview (see Figure 6) or a shopping scene with a father, his son, and a shopkeeper. With the student’s permission, these scenarios will be publicly available as examples that are part of the toolkit. From a software quality perspective, given the wide range of scenarios explored by the students, we were able to identify some issues with the toolkit, which were promptly fixed.

Fig. 6. Students' Job Interview Demo.

VII. Conclusion

In this paper, we argued that the development of serious games is faced with additional challenges that are related to the pedagogical goals that the designers have in mind. For instance, in games that are about teaching conversational skills, developers have to figure out how to offer a rich interaction space that supports the exploration and failure of different communicative actions and their associated socio-emotional effects.

In the mainstream gaming industry, dialogues are typically handled through branching structures that limit the set of possible interactions, by offering little flexibility in the way characters respond to what the players say to them. Alternatively, in the research field of virtual agents, researchers have developed and proposed tools for the creation of conversational agents that have rich socio-emotional models driving their behavior. These agents have great potential for being applied in serious games that teach soft skills, as their behaviors are more procedural and less scripted. However, so far, agent architectures are still far from being widely used in the serious games industry due to, in large part, accessibility issues. With those issues in mind, we took an existing virtual agent architecture, FAtiMA Modular, and adapted it to a new toolkit with the goal of making it more appealing to game developers. For that effect, we adopted a functional library pattern instead of a framework-based approach. Moreover, the functionality was divided in two main components, the Role-Play Character and the Integrated Authoring Tool. The first is responsible for managing the character's beliefs, memories and emotional state as well as running a decision-making process for each character according to its ascribed role. The second component allows the developer to manage the list of all the characters that are available in each game scenario as well as the available dialogues that the characters, including the player's avatar, can select from at any given state of the interaction.

The resulting toolkit was then applied successfully by a game studio, PlayGen, in the development of two serious games. The first game was designed to teach players how to properly communicate with emotional customers in a customer service setting. The second game has the player managing a sport sails team composed by multiple characters with different role preferences. Both of these games benefited from the use of the toolkit in adding emotional dynamics to their characters that is reflected in their decisions. Additionally, a group of 68 students successfully developed projects for a Game AI course using the toolkit. This experience was a good stress test on the toolkit given the wide variety of scenarios explored by the students.

As future work, we plan to conduct more formal user study centered around the authoring capabilities of the toolkit. The main idea will be to have participants watch a video tutorial about how the toolkit works and then be instructed to change an existing game scenario according to a set of predefined goals. The feedback obtained will then be used to further improve the toolkit.

Acknowledgments

This work was supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UID/CEC/50021/2013 and has been partially funded by the EC H2020 project RAGE (Realising an Applied Gaming Eco-System) Grant agreement No 644187.

References

[1] R. Aylett, M. Vala, P. Sequeira, and A. Paiva, "Fearnot!–an emergent narrative approach to virtual dramas for anti-bullying education," in *International Conference on Virtual Storytelling*. Springer, 2007, pp. 202–205.

[2] M. Cavazza, F. Charles, and S. J. Mead, "Character-based interactive storytelling," *IEEE Intelligent systems*, vol. 17, no. 4, pp. 17–24, 2002.

[3] F. Correia, P. Alves-Oliveira, N. Maia, T. Ribeiro, S. Petisca, F. S. Melo, and A. Paiva, "Just follow the suit! trust in human-robot interactions during card game playing," in *Robot and Human Interactive Communication (RO-MAN), 2016 25th IEEE International Symposium on*. IEEE, 2016, pp. 507–512.

[4] N. Degens and G. Hofstede, "Traveller - Intercultural training with intelligent agents for young adults," *Proceedings of the ...*, 2013.

[5] J. Dias, S. Mascarenhas, and A. Paiva, "Fatima modular: Towards an agent architecture with a generic appraisal framework." *Emotion Modeling*, vol. 8750, pp. 44–56, 2014.

[6] J. Dias and A. Paiva, "Feeling and reasoning: A computational model for emotional characters," in *EPIA*, vol. 3808. Springer, 2005, pp. 127–140.

[7] T. Economist, "Why video games are so expensive to develop," *The Economist Group Limited*, 2014. [Online]. Available: http://www.economist.com/blogs/economist-explains/2014/09/economist-explains-15

[8] J. Gratch, J. Rickel, E. André, J. Cassell, E. Petajan, and N. Badler, "Creating interactive virtual humans: Some assembly required," *IEEE Intelligent systems*, vol. 17, no. 4, pp. 54–63, 2002.

[9] W. L. Johnson and A. Valente, "Tactical language and culture training systems: Using artificial intelligence to teach foreign languages and cultures." in *AAAI*, 2008, pp. 1632–1639.

[10] F. Khatib, S. Cooper, M. D. Tyka, K. Xu, I. Makedon, Z. Popovic, D. Baker, and F. Players, "From the Cover: Algorithm discovery by protein folding game players," *Proceedings of the National Academy of Sciences*, vol. 108, no. 47, pp. 18 949–18 953, 2011.

[11] J. M. Kim, R. W. Hill Jr, P. J. Durlach, H. C. Lane, E. Forbell, M. Core, S. Marsella, D. Pynadath, and J. Hart, "Bilat: A game-based environment for practicing negotiation in a cultural context," *International Journal of Artificial Intelligence in Education*, vol. 19, no. 3, pp. 289–308, 2009.

[12] G. Koo and S. Seider, "Video Games for Prosocial Learning," *Ethics and Game Design*, pp. 16–33, 2010. [Online]. Available: http://128.197.153.21/seider/Consolidated papers/Prosocial Learning in Video Games Final Version_Ko.pdf%5Cnhttp://services.igi-global.com/resolvedoi/resolve.aspx?doi=10.4018/978-1-61520-845-6.ch002

[13] A. Martens, H. Diener, and S. Malo, "Game-based learning with computers–learning, simulations, and games," *Transactions on edutainment I*, pp. 172–190, 2008.

[14] M. Ninaus, K. Kiili, J. McMullen, and K. Moeller, "A Game-Based Approach to Examining Students' Conceptual Knowledge of Fractions," in *Games and Learning Alliance: 5th International Conference, GALA 2016, Utrecht, The Netherlands, December 5–7, 2016, Proceedings*, R. Bottino, J. Jeuring, and R. C. Veltkamp, Eds. Cham: Springer International Publishing, 2016, pp. 37–49. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-50182-6_4

[15] N. Padilla-Zea, F. L. Gutirrez, J. R. Lpez-Arcos, A. Abad-Arranz, and P. Paderewski, "Modeling storytelling to be used in educational video games," *Computers in Human Behavior*, vol. 31, pp. 461 – 474, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0747563213001295

[16] P. Sajjadi, E. El Sayed, and O. De Troyer, "On the Impact of the Dominant Intelligences of Players on Learning Outcome and Game Experience in Educational Games: The TrueBiters Case," in *Games and Learning Alliance: 5th International Conference, GALA 2016, Utrecht, The Netherlands, December 5–7, 2016, Proceedings*, R. Bottino, J. Jeuring, and R. C. Veltkamp, Eds. Cham: Springer International Publishing, 2016, pp. 221–231. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-50182-6_20

[17] A. Santos, F. Strada, and A. Bottino, "Games and Learning Alliance," in *Games and Learning Alliance: 5th International Conference, GALA 2016, Utrecht, The Netherlands, December 5–7, 2016, Proceedings*, ser. Lecture Notes in Computer Science, A. De Gloria, Ed. Cham: Springer International Publishing, 2015, vol. 9221, no. April 2016, pp. 73–82. [Online]. Available: http://link.springer.com/10.1007/978-3-319-22960-7

[18] U. Spierling and N. Szilas, "Authoring issues beyond tools," in *Joint International Conference on Interactive Digital Storytelling*. Springer, 2009, pp. 50–61.

[19] W. Van Der Vegt, W. Westera, E. Nyamsuren, A. Georgiev, and I. M. Ortiz, "RAGE Architecture for Reusable Serious Gaming Technology Components," *International Journal of Computer Games Technology*, vol. 2016, 2016.

[20] N. Vannini, S. Watson, K. Dautenhahn, S. Enz, M. Sapouna, D. Wolke, S. Woods, L. Hall, A. Paiva, E. André, R. Aylett, and W. Schneider, ""FearNot!": A computer-based anti-bullying-programme designed to foster peer intervention," *European Journal of Psychology of Education*, vol. 26, no. 1, pp. 21–44, 2011.

[21] L. von Ahn and L. Dabbish, "Labeling images with a computer game," *Proceedings of the 2004 conference on Human factors in computing systems - CHI '04*, pp. 319–326, 2004. [Online]. Available: http://portal.acm.org/citation.cfm?doid=985692.985733

The Influence of Feedback Choice on University Students' Revision Choices and Performance in a Digital Assessment Game

Maria Cutumisu
Department of Educational Psychology
University of Alberta
Edmonton, Canada
cutumisu@ualberta.ca

***Abstract*— This study examines the effect of choosing versus being assigned critical feedback on performance and on the choice to revise digital posters. Participants were North American University students (n = 125) randomly assigned to one of two conditions, Choose and Receive. In both conditions, participants designed three posters in Posterlet, an assessment game with an embedded intelligent feedback system that assessed their learning processes and poster performance. In a yoked experimental study design, participants in the Choose condition (n = 71) could choose to receive either critical or confirmatory feedback, while participants in the Receive condition (n = 54) were assigned the same amount and order of critical feedback chosen by their Choose condition counterparts. Results revealed no differences between conditions in performance and students' choices to revise their posters. Implications for designing intelligent feedback-rich learning and assessment environments are discussed.**

***Index Terms*—intelligent feedback, choice, assessment game, performance, revision**

I. INTRODUCTION

This study explores the impact of feedback-seeking choice on students' performance and on their learning behaviours (i.e., choice to revise their artifacts). This topic is of increased relevance, as more and more emphasis in education is placed on pedagogical approaches such as personalized learning, as well as on differentiated instruction and attention to students. These approaches aim to "increase learner choice and voice" [1].

Thus, this research has an immediate impact on designing curricula that increase learners' locus of control, enabling students to make more decisions about their learning. Despite this, most research focuses on situations in which feedback is assigned to students, rather than on situations in which students proactively seek feedback. The latter is a more naturalistic scenario to examine, as the ability to recognize when to seek feedback and what type of feedback is most useful is essential for supporting independent, innovative learners.

As feedback fills a gap between what learners know and what they desire to know, there are reasons to believe that critical (constructive) feedback influences learning, but there is no clear evidence that the choice of critical feedback is more important than simply assigning critical feedback to learners as a result of their performance. However, individuals often need to seek critical feedback to improve their performance or to learn a new concept [2, 3], especially as most new ideas need critical constructive feedback to become successful [4].

The mechanisms of critical feedback seeking and processing are not yet understood, as research scrutinizing the link between feedback and performance has yielded mixed results [5, 6, 7]. A study found that novices sought confirmatory (positive) feedback more often, whereas experts sought critical (negative) feedback more often [8], but performance was not measured.

Other factors, such as mindset, may interfere with both the decision to choose critical feedback and the heeding of critical feedback [9-15]. Another important aspect of delivering feedback is timing. In most cases, immediate feedback can improve learning. For instance, peer feedback delivered within 24 hours in a massive open online course was more effective for learning than feedback delivered after 24 hours [16].

Previous research showed that choosing critical feedback was associated with better performance and more revisions [18]. The current study explores if this result persists when students are being assigned critical feedback instead of choosing it.

This experiment was designed to elucidate this matter by answering three research questions:

1) Do learning behaviours correlate with performance by condition?

2) Does condition moderate the relation between learning behaviours and performance?

3) Are there any outcome differences between choosing and receiving feedback?

978-1-5386-4359-4/18/$31.00 ©2018 IEEE

II. METHOD

A. Participants, Procedure, and Data Sources

Participants were n = 125 students from the Faculty of Education Research Participation Pool Program at a large North American university (see Table 1). They were informed about the study and they provided online informed consent for their participation. Participants were randomly assigned to one of two conditions, Choose and Receive.

TABLE I. PARTICIPANT AND STUDY INFORMATION

Cond.	Gender		Total	M_{age} (SD) in Years	$M_{duration}$ (SD) in Minutes
	Female	*Male*			
Choose	44	27	71	24.48 (5.03)	11.39 (5.41)
Receive	31	23	54	25.54 (6.04)	10.41 (4.03)
Total	75	50	125	24.94 (5.49)	10.97 (4.87)

Two versions of an online game assessment instrument, in which students designed posters and learned graphic design principles from feedback, were employed to collect data. In the Choose condition (see Figure 1), the player clicks on one box ("I like" or "I don't like") above each character to choose either confirmatory or critical feedback.

Fig. 1. In the Choose condition, the player chose critical feedback from the lion and then confirmatory feedback from the elephant. (Reprinted from Cutumisu & Schwartz, 2018.)

In the Receive condition (see Figure 2), the player clicks on the "Click for feedback" box to reveal feedback of a valence that is assigned by the game. According to a one-to-one yoked experimental design protocol, each Receive condition participant was assigned the same amount and order of critical feedback as that chosen by a matched Choose condition participant.

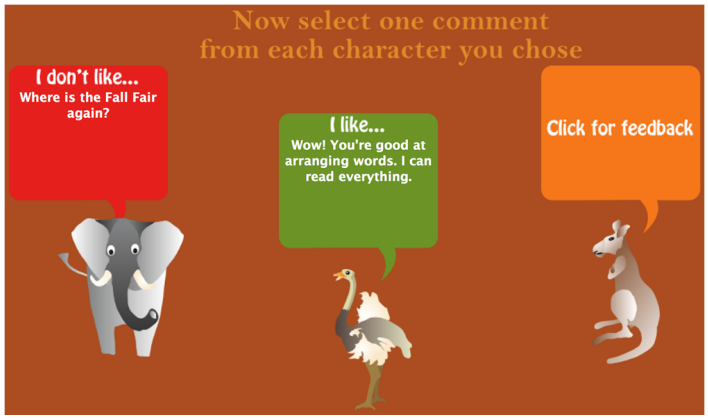


Fig. 2. In the Receive condition, the player first clicked on the elephant and received critical feedback, then on the ostrich and received confirmatory feedback. (Reprinted from Cutumisu & Schwartz, 2018.)

In contrast to the previous correlational research conducted using Posterlet, the current research proposes an experiment based on the original Posterlet version where students chose their feedback (Figure 1) and our modified version where students are assigned feedback (Figure 2).

B. The Posterlet Assessment Instrument

This experimental study employed the Posterlet assessment instrument [18]. Posterlet is an assessment game that has an intelligent feedback system which parses each digital poster and produces a performance score and a set of feedback phrases for each of the graphic design principles used correctly as well as incorrectly on that poster. The feedback system also keeps a trail of which principles were used on each of the three posters and determines dynamically the kind of feedback message to display to each student depending on their current performance and the performance on the previous posters across the game.

The game's feedback system generates feedback according to a priority scheme that is based on three broad categories of graphic design principles: crucial information (e.g., the date, time, and location of the poster fair need to be included on the poster), readability (e.g., the contrast between the colour of the poster canvas and the colour of the text must be high), and space use (e.g., the text must not be placed too close to the edge of the poster). The feedback system generates feedback by selecting feedback messages successively from these categories [19].

In the Posterlet game, students follow the steps illustrated in Figure 3 three times. Each time, they choose a booth (Step 1) and design a poster for it (Step 2). Then, they select three virtual animal characters from a focus group (Step 3), choose either critical or confirmatory feedback from each of them about the poster and read the feedback (Steps 4 and 5). Then, they choose to revise (Step 6) or submit (Step 7) their poster. Posterlet tracks the number of critical feedback and revisions students make, and it computes an overall poster performance.

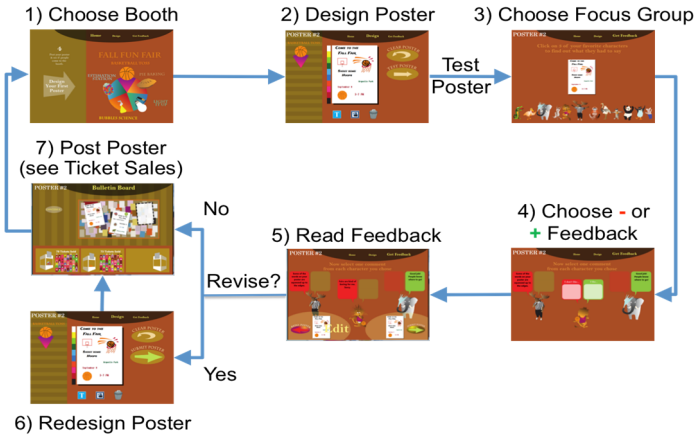


Fig. 3. The Posterlet game flow used to create three posters. (Reprinted from Cutumisu, Blair, Chin, & Schwartz, 2015.)

C. Measures

The study focuses on three main classes of measures: behaviours (Critical Feedback and Revision), poster performance (Poster Quality), and condition (Choose versus Receive). These measures together with the relations among them are illustrated in Figure 4.

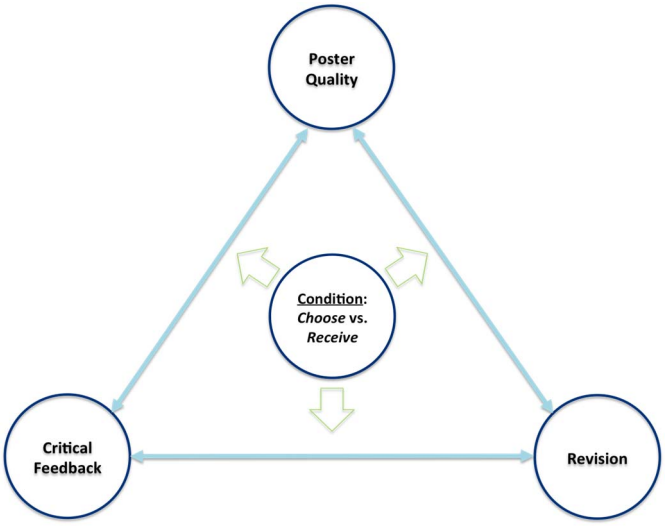


Fig. 4. The three classes of measures employed in the study. The outer arrows indicate associations between measures, while the inner arrows indicate whether the experimental condition moderates the relations between measures. Solid arrows indicate significant relations.

1) In-game learning behaviours

Critical Feedback measures the number of times students encountered (i.e., chose or received) critical feedback (i.e., "I don't like…"). As there were three posters with three pieces of feedback per poster, this measure ranged from zero to nine.

Revision measures the number of posters a student chose to revise. As there were three posters with one opportunity to revise per poster, this measure ranged from zero to three.

2) In-game performance

Poster Quality measures students' performance based on 21 design principles summed across the game. The quality of each poster is the sum of 21 feature scores: 1 if a feature is always used correctly on a poster, 0 if not included, and -1 if used incorrectly on a poster.

Pretest measures the quality of the first poster, before revision.

III. Results

This section describes the results of the statistical analyses conducted to answer the three main research questions of this study.

A. Do learning behaviours correlate with performance by condition?

Spearman correlations were conducted per condition between performance and behaviours (critical feedback and revision), as these variables were not normally distributed. The results of these analyses are shown in Table 2 and Table 3. In both conditions, Critical Feedback correlated with Poster Quality and with Revision. A t-test analysis comparing the strength of the correlation coefficients between conditions [20, 21] indicated that Critical Feedback and Revision were associated significantly stronger in the Choose than in the Receive condition (z-score = 2.92, $p < .01$).

Moreover, Poster Quality correlated with Revision only in the Choose condition. This suggests that the experimental condition may moderate the relation between poster quality and the choice to revise, as well as between the two learning behaviours (choosing or receiving critical feedback and revising). As the choice to revise follows students' interaction with feedback, this suggests an interaction of critical feedback with revision to predict performance. The next research questions aims to test this hypothesis.

TABLE II. Correlations between learning behaviours and performance for the *Choose* condition (**$p < .01$, *$p < .05$)

Measures (n = 71)	Revision	Poster Quality
Critical Feedback	**.68****	**.29***
Revision	--	**.37****

TABLE III. Correlations between learning behaviours and performance for the *Receive* condition (**$p < .01$, *$p < .05$)

Measures (n = 54)	Revision	Poster Quality
Critical Feedback	**.29***	**.33***
Revision	--	.17

A two-way repeated measures analysis of variance showed a significant growth of Poster Quality [$F(1.79, 219.73) = 31.51$, $p < .001$, $\eta^2 = .20$]. As participants improved their poster

performance across the game, Poster Quality was considered a learning measure, as shown in Figure 5.

There was no main effect for condition [$F(1, 123) = .98$, $p = .32$, $\eta^2 = .008$] and no significant interaction between condition (although Choose participants achieve slightly better performance than Receive participants) and poster levels [$F(1.79, 219.73) = .33$, $p = .70$, $\eta^2 = .003$]. In the *Choose* condition, participants improved their average performance from $round_1 = 11.41$ (SD = 4.07) to $round_2 = 13.70$ (SD = 3.91) to $round_3 = 13.97$ (SD = 3.41). In the *Receive* condition, participants also improved their average poster performance from $round_1 = 10.76$ (SD = 4.88) to $round_2 = 12.85$ (SD = 4.07) to $round_3 = 13.70$ (SD=4.03). Moreover, this growth did not vary by condition.

Post Hoc tests revealed that, for both conditions, there was a significant increase in poster quality from $round_1$ to $round_2$ and $round_3$, but the increase from $round_2$ to $round_3$ was significant only for the Receive condition.

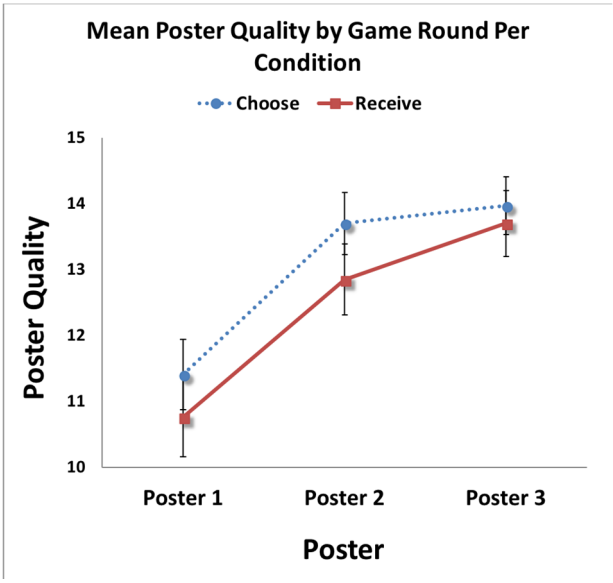


Fig. 5. A two-way repeated measures analysis of variance used the independent categorical variable Poster (with three levels that correspond to the three posters) and condition (Choose versus Receive) to predict the dependent variable, Poster Quality. Results showed a significant growth of Poster Quality, but no main effect for condition and no significant interaction between the experimental condition and poster levels. The increase from $round_2$ to $round_3$ was significant only for the Receive condition.

Finally, given the association of learning behaviours with performance, standard multiple regression analyses were conducted to investigate whether Critical Feedback and Revision were independent predictors of performance (Poster Quality) in each condition.

In the *Choose* condition, the model composed of the two learning behaviours, Critical Feedback and Revision, predicted Poster Quality significantly [$F(2,68) = 4.94$, $p < .05$, $R^2 = .13$, Adjusted $R^2 = .10$] and Revision [Beta = .33, B = 2.67, SE = 1.22, $t = 2.19$, $p = .03$] was a significant predictor, but Critical Feedback [Beta = .03, B = .18, SE = .79, $t = .22$, $p = .82$] was not a significant predictor.

In the *Receive* condition, the model was significant [$F(2,51) = 4.29$, $p < .05$, $R^2 = .14$, Adjusted $R^2 = .11$]. Moreover, Critical Feedback was a significant predictor: Beta = .29, B = 1.65, SE = .77, $t = 2.12$, $p = .04$, but Revision was not: Beta = .17, B = 1.72, SE = 1.41, $t = 1.22$, $p = .23$.

B. Does condition moderate the relation between learning behaviours and performance?

Because revision occurs after receiving feedback, it seemed possible that critical feedback would determine whether students chose to revise due to constructive criticism. To find out whether the experimental condition interacts with critical feedback in predicting the choice to revise, we conducted two-way analyses of variance.

Participants were divided into two percentile groups according to the amount of critical feedback they encountered (Group 1: zero to five pieces of critical feedback; Group 2: six to nine pieces of critical feedback).

Then, critical feedback and condition were used as independent categorical variables to predict the dependent variable, Revision.

Results yielded a significant interaction between the Critical Feedback Group and the experimental condition: $F(1,121) = 10.41$, $p < .01$, $\eta^2 = .08$. Specifically, participants who encountered more critical feedback also chose to revise significantly more but only in the Choose condition (see Figure 6 and Figure 7).

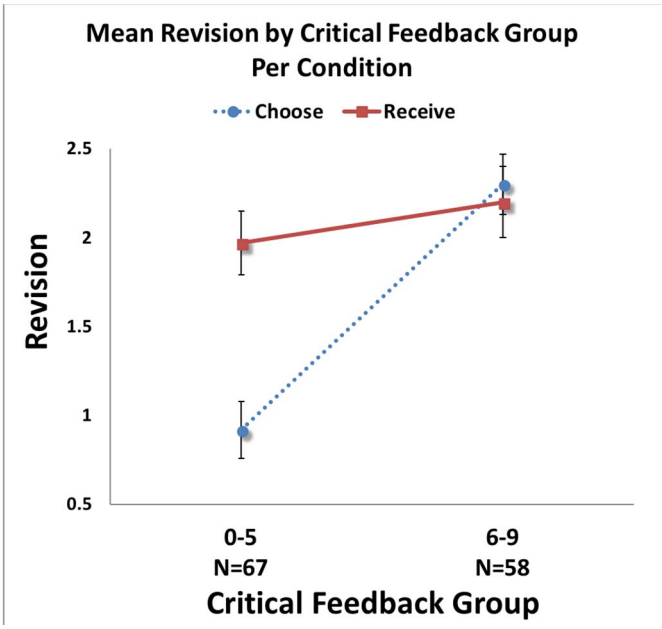


Fig. 6. A two-way analysis of variance used the independent categorical variables Critical Feedback Group (low versus high levels) and condition (Choose versus Receive) to predict the dependent variable, Revision. A significant interaction between Critical Feedback Group and condition showed that students who encountered more critical feedback in the game also chose to revise significantly more, but only in the Choose condition.

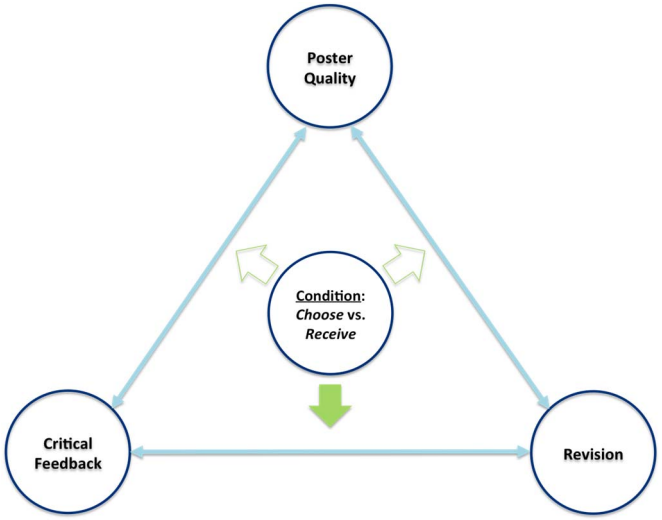


Fig. 7. Condition moderates only the relation between critical feedback and revision (i.e., solid inner arrow pointing from Condition to the relation between Critical Feedback and Revision). This reveals that students who encounter high levels of critical feedback in the game also revise more, but only in the Receive condition (i.e., only when they are assigned their feedback).

Finally, two-way analyses of variance examined the effect of condition (Choose versus Receive) and behaviours (critical feedback levels and revision levels, respectively) on poster performance, controlling for the pretest.

Participants were divided into two percentile groups according to the amount of revisions they made (Group 1: zero or one revision; Group 2: two or three revisions).

Results revealed a main effect for pretest (i.e., students who revise more often also design better posters), but no main effect for critical feedback or condition, and no interactions of critical feedback and condition for performance.

Similar results yielded for revision and condition predicting performance (controlling for the pretest). This shows that condition does not moderate the relations between behaviours and performance, as illustrated by the two empty inner arrows of Figure 7.

C. Are there any outcome differences between choosing and receiving feedback?

Non-parametric Mann-Whitney independent-samples tests were conducted to determine whether there were any learning outcome differences between the two groups (i.e., participants who had a choice over feedback and those who did not).

Figure 8 and Figure 9 illustrate each outcome measure computed across the game as a function of critical feedback (from zero to nine) by condition. The x-axis shows the range of critical feedback.

Analyses revealed no differences in Poster Quality (Mean $Rank_{Choose} = 65.53$, Mean $Rank_{Receive} = 59.68$, $Z = -.90$, $p = .37$; see Figure 8) but significant differences in Revision (Mean $Rank_{Choose} = 56.44$, Mean $Rank_{Receive} = 71.62$, $Z = -2.43$, $p < .05$; see Figure 9) between conditions. Thus, students in the Receive condition revised significantly more than students in the Choose condition.

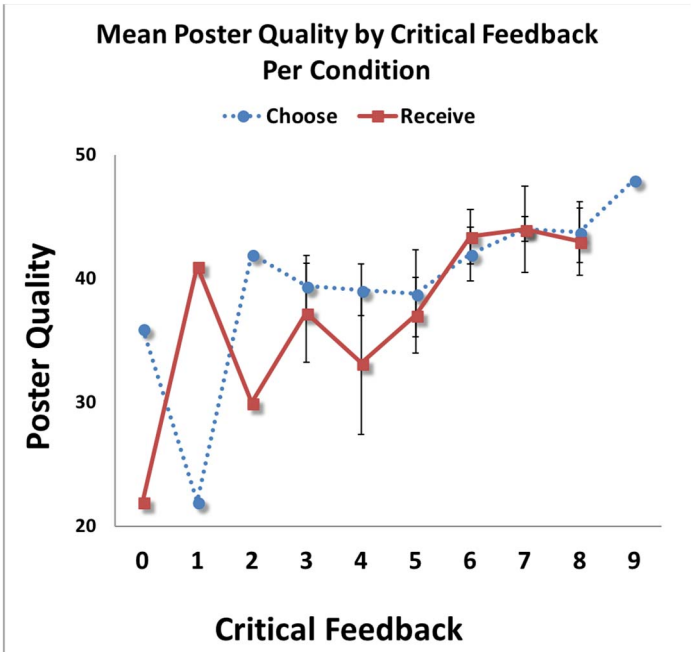


Fig. 8. Poster performance as measured by Poster Quality for each level of critical feedback by condition (Choose versus Receive). The Choose condition is represented by a blue dotted line, while the Receive condition is represented as a solid red line. Error bars represent +/- one standard error.

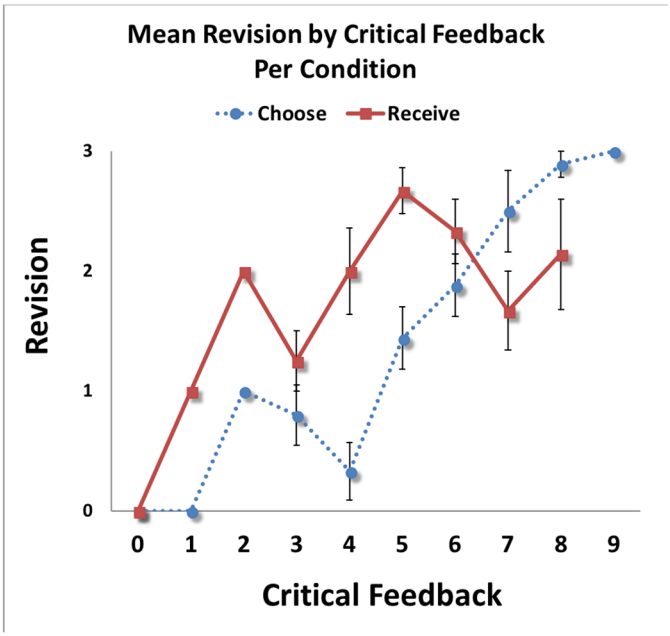


Fig. 9. Revision choices for each level of critical feedback by condition (Choose versus Receive). The Choose condition (i.e., students choose between critical and confirmatory feedback following the design of a poster) is represented by a dotted blue line, while the Receive condition (i.e., students are assigned a schedule of critical or confirmatory feedback following the design of a poster) is represented as a solid red line. Error bars represent +/- one standard error.

IV. Discussion and Significance

A. Do learning behaviours correlate with performance by condition?

The associations between learning behaviours and performance were similar in the two conditions. However, the relation between learning behaviours was stronger in the Choose than in the Receive condition. This may be due to a motivational factor related to having a choice over one's learning.

Moreover, in the Choose condition, Revision was more important than Critical Feedback for performance. However, in the Receive condition, this situation was reversed (i.e., Critical Feedback was more important than Revision in predicting performance).

To date, this is a first demonstration that choosing and receiving critical feedback show similar patterns of influencing performance (students learn at a similar pace regardless of condition, as their poster performance improved gradually), but that condition influences the relation between learning behaviours (critical feedback and revision), corroborated by the mediation analyses of the next section.

These findings have multiple implications for personalized learning curricula that may consider choice as a behaviour motivator that would not affect the learners' performance, as we found that feedback valence choice did not make a difference in students' performance.

B. Does condition moderate the relation between learning behaviours and performance?

Analyses examining interactions between condition and choices to predict performance indicated that condition was not a moderator of these relations. However, condition moderated the relation between critical feedback and revision, showing that different levels of critical feedback did not make a difference for the revision behaviour of students in the Receive condition, but it did for students in the Choose condition who revised significantly more when they chose higher levels of critical feedback.

This finding supports the importance of choice in nudging students' revision behaviours given an appropriate level of critical feedback encountered after solving a task (e.g., poster design in the case of Posterlet).

C. Are there any outcome differences between choosing and receiving feedback?

Results showed no differences between conditions in performance, except for differences in Revision. Students in the Receive condition revised more than their Choose condition counterparts. This finding supports the previous two results. It also suggests that students are likely to revise more when they are assigned their feedback valence.

Consequently, this suggests that when students are assigned critical feedback without exercising a choice about their feedback valence, they may revise more not because of exercising their usual learning behaviours, but because of reconciling their expectations with the unwelcomed feedback message, as supported by prior research [17, 18].

This experiment examined the associations between learning behaviours (critical feedback and revising) and performance in two different conditions (choosing the valence of the feedback versus being assigned that feedback valence). The yoked study design enables a better understanding of the processes unfolding when students interact with critical feedback by controlling for the decision to seek critical feedback.

Moreover, as the experiment demonstrated that performance is not related to students' decision to seek critical feedback but to simply encountering more critical feedback, future research can focus on discovering why some students seek critical feedback more often than others and why they decide to revise their work more often.

In the near future, this research will investigate the effect of choosing versus receiving critical feedback on learning and mindset, as well as the effect of choosing versus receiving critical feedback on students' memory for critical feedback, to gain an insight into the underpinnings of critical feedback processing for performance improvement.

These results have implications for designing instructional materials and ways of delivering feedback. Findings suggest that personalized learning would benefit from a focus on the content, the amount, and the valence of feedback, more than on the feedback choices that students exercise in learning environments.

V. Conclusions and Educational Implications

This experiment constitutes a novel empirical examination of the impact of both the agency (choosing versus receiving) and the valence (critical versus confirmatory) of feedback on performance. Results suggest that critical feedback is associated with better performance and increased willingness to revise digital posters in a game-based behavioural assessment, regardless of being chosen or assigned.

Thus, the design of feedback-rich learning and assessment environments may benefit from facilitating students' interaction with higher levels of critical feedback with the aim to improve student performance and learning behaviours (e.g., frequency of revising their work), regardless of whether students choose critical feedback or receive it.

Acknowledgment

I would like to express my gratitude to the students who participated in this research, to the Research Participation Pool program, as well as to the following granting agencies for supporting this research: the University of Alberta *Support for the Advancement of Scholarship* Grant and the *Social Sciences and Humanities Research Council of Canada (SSHRC) Insight Development Grant (IDG)* # RES0034954.

I would also like to thank Dr. Daniel L. Schwartz and the AAA Lab team members at the Stanford Graduate School of Education for their support with using the two versions of the Posterlet computer game-based assessment in this experiment.

REFERENCES

[1] T. Rudd, "Learning spaces and personalisation workshop outcomes". Bristol, UK: NESTA Futurelab. Retrieved in June 2018 from http://www.futurelab.org.uk/resources/documents/event_presentations/Learning_Spaces_and_Personalisation_workshop.pdf.

[2] E. F. Williams and J. Ehrlinger, "Failing to learn from feedback: Inter- and intrapersonal roadblocks to autonomous learning," Autonomous Learning in the Workplace, 263-286, 2017.

[3] B. J. Zimmerman, "Self-regulated learning and academic achievement: An overview," Educational Psychologist, 25(1), 3-17, 1990.

[4] J. G. March, Explorations in Organizations. Stanford University Press, 2008.

[5] J. Hattie and H. Timperley, "The power of feedback," Review of Educational Research, 77(1), 81-112, 2007. DOI=10.3102/003465430298487.

[6] A. N. Kluger and A. DeNisi, "The effects of feedback interventions on performance: A historical review, a meta-analysis, and a preliminary feedback intervention theory," Psychological bulletin, 119(2), 254, 1996.

[7] A. N. Kluger and A. DeNisi, "Feedback interventions: Toward the understanding of a double-edged sword," Current Directions in Psychological Science, 7(3), 67-72, 1998.

[8] S. R. Finkelstein and A. Fishbach, "Tell me what I did wrong: Experts seek and respond to negative feedback," Journal of Consumer Research, 39(1), 22-38, 2012.

[9] C. S. Dweck, Self-theories: Their Role in Motivation, Personality, and Development. Philadelphia, PA: Psychology Press/Taylor and Francis, 1999.

[10] C. S. Dweck, "The Journey to Children's Mindsets—and Beyond," Child Development Perspectives, 11(2), 139-144, 2017.

[11] C. S. Dweck and E. L. Leggett, "A social-cognitive approach to motivation and personality," Psychological Review, 95(2), 256-273, 1988.

[12] J. Ehrlinger, A. L. Mitchum, and C. S. Dweck, "Understanding overconfidence: Theories of intelligence, preferential attention, and distorted self-assessment," Journal of Experimental Social Psychology, 63, 94-100, 2016.

[13] J. A. Mangels, B. Butterfield, J. Lamb, C. Good, and C. S. Dweck, "Why do beliefs about intelligence influence learning success? A social cognitive neuroscience model," Social Cognitive and Affective Neuroscience, 1(2), 75-86, 2006.

[14] A. J. Martin, "Implicit theories about intelligence and growth (personal best) goals: Exploring reciprocal relationships," British Journal of Educational Psychology, 85(2), 207-223, 2015.

[15] M. Zingoni and K. Byron, "How beliefs about the self influence perceptions of negative feedback and subsequent effort and learning," Organizational Behavior and Human Decision Processes, 139, 50-62, 2017.

[16] C. E. Kulkarni, M. S. Bernstein, and S. R. Klemmer, "PeerStudio: rapid peer feedback emphasizes revision and improves performance," In Proceedings of the Second (2015) ACM Conference on Learning@ Scale, 75-84. ACM, 2015.

[17] M. Cutumisu, and D. L. Schwartz, "The impact of critical feedback choice on students' revision, performance, learning, and memory," Computers in Human Behavior, 78, 351-367, ISSN 0747-5632, 2018.

[18] M. Cutumisu, K. P. Blair, D. B. Chin, and D. L. Schwartz, "Posterlet: A game-based assessment of children's choices to seek feedback and to revise," Journal of Learning Analytics, 2(1), pp. 49-71, 2015.

[19] M. Cutumisu, K. P. Blair, D. B. Chin, and D. L. Schwartz, "Assessing whether students seek constructive criticism: The design of an automated feedback system for a graphic design task," International Journal of Artificial Intelligence in Education (IJAIED), 27(3), 419-447, DOI: 10.1007/s40593-016-0137-5, Springer, 2016.

[20] R. A. Fisher, "On the probable error of a coefficient of correlation deduced from a small sample," Metron, 1, 3-32, 1921.

[21] D. Soper, "Significance of the difference between two correlations calculator [Software]". Retrieved in June 2018 from http://www.danielsoper.com/statcalc.

A Plot from the Stars: Educational Game Development for Teaching Basic Mathematical Functions

Gabriel Toschi de Oliveira,
Claudio Fabiano Motta Toledo,
Seiji Isotani and Geiser Chaclo Challco
Institute of Mathematics and Computer Science
University of Sao Paulo, Sao Carlos, Brazil
gabriel.toschi.oliveira@usp.br, claudio@icmc.usp.br,
sisotani@icmc.usp.br,geiser@usp.br

Hugo Henriques Pereira
Savannah College of Art and Design
Savannah, EUA
Hperei20@student.scad.edu

***Abstract*—The digital games are consolidating as one of the biggest entertainment industries in the latest years, as well as their use for educational applications. However, some educational games do not provide satisfactorily a pleasurable and engaging game experience. This issue compromises the learning process of the content approached by the game. This paper introduces the puzzle game "A Plot from the Stars", whose goal is to incorporate basic mathematical function concepts through a fun gameplay. During the game, the player has the role of a cadet within a space station and must use mathematical graphs as laser beams to destroy enemy ships. The first version of the game is introduced here as well as the results achieved after its initial evaluation by a group of students.**

***Index Terms* — serious games, educational games, mathematical function, learning process.**

I. Introduction

The digital game market has grown a lot since its inception in the 1960s. The first electronic game in history, Spacewar, was made just as a student experience at one of the MIT labs in 1961 [1]. More than 50 years later, the video game industry is already surpassing the film and music industries [2] in values involved. The consumers spent a total of $30.4 billion on electronic games in the United States [3] during 2016.

The use of electronic games as an educational tool is recent and it was leveraged by the emergence of modern and affordable gaming development tools. The electronic gaming appears as an alternative to develop new learning opportunities [4]. The present paper introduces the educational computer game "A Plot from the Stars", which is composed by a series of logical challenges. To complete all the puzzle levels successfully, the student has to learn the correlation between different mathematical functions and their graphs.

Our approach aims to find a balance between a highly engaging and fun game as well as an educational tool that works effectively for student groups. Our contribution lies in the proposition of a suitable solution to this issue by making gameplay and educational content completing each other. The basic concepts of the gameplay are described in this paper to show precisely how the game works. The Unity game engine was chosen since it is widely used in the development of professional games [5].

Finally, we report preliminary results achieve evaluating the first version of our game by a set of players. This is done using strategies such as pre-test, post-test and IMI (Intrinsic Motivation Inventory). These results were collected from a beta version of the game with a reduced set of levels. However, the set of levels is sufficient to represent and evaluate the main concepts of the proposed game.

The paper is organized as follows. Section II reviews some related works, while the gameplay concepts is described in section III. The results achieved from a beta version of the game are reported in section IV. The conclusions are in section V.

II. Related Work

In this section, we present some works related to the use of electronic games as an educational tool over the last few years, as well as some important titles in the area of educational games. First, we will focus on a brief history of educational game evolution. One of the first games to stand out was "The Oregon Trail" in 1971. The game sought to teach content from the history of the United States, specifically about the life of American settlers in the mid-nineteenth century. It became a major success in classrooms across the country at the time of launch [6].

In the early 1980s, the popularization of personal computers facilitated the growth of the educational gaming market. One of the main icons of this growth was the game series with the character Carmen Sandiego, a woman wanted for various crimes. In her debut game, "Where in the World is Carmen Sandiego", the player had a series of missions to accomplish and the main clues to unravel the mysteries were the player geographical knowledge about countries of the world [7].

The computers were a great home for electronic educational games success until the late 20th century with other big names such as the “Reader Rabbit“ series, produced by The Learning Company, and the game “Math Blaster“, developed by Davidson [7]. The latter was the first electronic games to have prominence in the teaching of Mathematics. However, its main mechanics relied only on solving a series of equations by shooting at generic targets [8]. These targets appears on the screen with numbers attached to them, without effectively introducing educational concepts during the gaming experience.

The market begins to decline for educational games in 2000 due to the lack of new products and innovative experiences on the side of developers [7]. This market has remained modest since then and educational games for mobile applications arise as a contemporary alternative. The authors in [9], [10] and [11] discuss the educational approach using classroom tablets.

The effectiveness of educational electronic games was already subject of discussion even when this market was at its peak. The work in [8] evaluates several big companies games already cited as ”drill and practice” software. In such approaches, the authors in [8] argue that teaching does not become fun or attractive. It is only done in a traditional and ”sweetened” way with failed game mechanics. However, the author also believes that ”teaching can be fun on its own” and argues favorably to this approach with the use of educational games. This concept is one of the pillars for the development of ”A Plot from the Stars”.

One of the most addressed areas in the development of educational electronic games is Mathematics, since the beginning of this market, as the aforementioned “Math Blaster“. Another contemporary game, “Logical Journey of the Zoombinis“, brings interesting mechanics based on mathematical logic, combinations and arrangements of data and function graphs. The game provides a playful and intriguing environment with relevant use of feedbacks from the player during matches [12]. These playful and feedback aspects are also present in the online adventure game “Lure of the Labyrinth“, which uses interactive and interesting puzzles to teach basic math concepts such as proportions, quantities, fractions and distance [13].

The educational approaches shown in [12] and [13] present a desired balance between fun and teaching, but they do this through small mini-games with simpler and different mechanics between each level. Our goal with ”A Plot from the Stars” is to expand similar concepts of game design producing complex and solid mechanics. The idea is to allow a robust and complete experience to the player with an efficient use of the mathematical contents.

Another educational approach can be seen in games like “Decimal Point: The Fantastically Fabulous World of Fractional Fun“, whose focus is to introduce concepts about decimal numbers to elementary school students [14]. Unlike the previous works mentioned and the game proposed by our paper, ”Decimal Point” uses a generic way to introduce educational concepts, bringing them only in the form of questions. It is similar to traditional methods of educational evaluation Our proposal follows another path by incorporating effectively contents into gameplay. Thus, we try to become gameplay and content two inseparable elements of game design, where ”learn” and ”play” are almost synonyms for the player.

Some works discuss possible approaches that can be used to help achieve the desired goal with this game design approach. Some concepts for improving the quality of educational electronic game to attract students are reported in [8]. These concepts talk about how to get the attention of the player and to help the learning in a beneficial way. On the other hand, a game design methodology is presented in [15] for the development of educational games. It consists of designing an educational game supported by three pillars: what contents will be addressed, what resources and mechanics will be used to address these contents and how the game will know that an effective learning is going on.

III. Gameplay Concepts

This section describes the concepts and mechanics of the game “A Plot from the Stars“, with focus on its main elements and how they are related to the learning of mathematical function.

A. Game Overview

The objective of the game is to complete a series of levels compounded by mathematical challenges. These levels have enemy spaceships that need to be destroyed to avoid the planet Earth invasion. The player must correctly position curves by using mathematical functions across the playing area. The changes in the coefficients of the mathematical functions will allow the player designing the best graph. Each graph plays the role of a high-energy light beam which is able to destroy the enemy spaceships. Figure 1 presents a typical view of a game level.

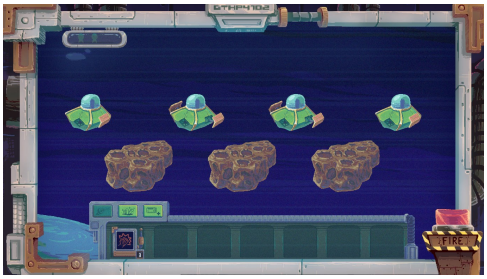


Fig. 1. Typical view of a ”A Plot from the Stars” level. In the bottom, the inventory with the satellite tab opened.

To position and calibrate these functions, the player has at her/his disposal an inventory of items that will be discussed in subsection III-B. Each level has a different inventory, allowing the game designers to control what information the player will have to solve a certain level. The aim is to encourage the player to learn specific features of the functions from the available information.

The level has asteroid figures which are specific spots to place graphs. These points are considered as the possible centers of Cartesian systems in which a graph can be drawn.

We choose to limit the positions where the graphics are positioned instead of allowing them to be placed in any position of the screen. This will allow the player to focus on specific elements, while the entire screen as an available option could divert the attention of the player. In addition, this technique allows the player not to ”break” levels, i.e., to find an easier or simpler solution for a level by eliminating the challenge present in the puzzle. Each Cartesian system present in an asteroid is shown only within a reduced domain, as can be seen in the Figure 2. Only those part of the graphs closest to the center of the system are shown, and the players have to combine different graphs to arrive at their solution to the level.

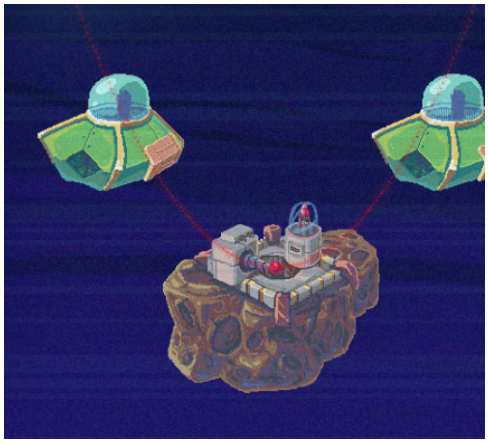


Fig. 2. A satellite with a Pharabulum Ruby attached, generating a parabola light beam.

The enemy ships can be destroyed (or not, if the player's solution is not valid) only when the Fire button is triggered, after the positioning of graphs. Before pushing the Fire button, a sketch of the graph is shown to the player providing a visual feedback to the student. At this moment, the student has the opportunity to evaluate how the graph will behave after changing coefficients in the function. This draft serves as a game element to help the player learning when looking for a valid solution.

Each level receives the title of ”mission” since it refers to the role of the player in the game universe. The player is a cadet in a space defense station and, for a mission to be completed successfully, all enemy ships must be destroyed without exception when pressing the Fire button. If the player achieves this goal, the next mission becomes available. However, if she/he wishes, the player can repeat missions that have already been completed. If the mission fails, the player has two options. The level can be completely by bringing the player to the original state. The other option is just retry to correct the previous solution. In this case, all ships return to their original positions, but the configuration of the previous solution remains unchanged to be improved.

B. Inventory and Items

All the manipulation of mathematical formulas, their corresponding graphs and the light beams governed by them are based on three types of items: satellites, crystals and modules. Each of them abstracts a different concept from the light beam, allowing the player to have a more precise and modular control of the mechanics. There is a finite and different collection of those type of items in each level, named as ”inventory.” This area of the game, shown in Figure 3, gives information about how the puzzle can possibly be solved, acting as a starting point when looking for solutions. Different inventories provide variability for levels and new challenges for players.

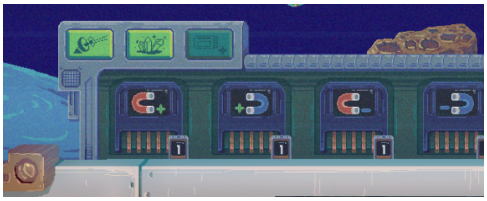


Fig. 3. The game inventory, with the module tab opened.

1) Satellites: A satellite, shown in Figure 2, represent a technology installed on an asteroid in the game universe, which allows light beams to be created. One of its main purposes within the gameplay is to become the player able to select the main characteristic of the light beam. The selected characteristics will define how the player will interact with the enemy ships and other screen elements.

For instance, the ”Attack Satellite” and the ”Attack Satellite EX”, both Attack-type satellites, give to the light beam the possibility of destroying all the common enemy ships touched by them. It is worth mentioning that the interaction of the light beam is independent of the mathematical function that generates its format. Thus, straight lines or parabolas can be used to destroy ships since generated from an Attack-type satellite. The separation between light beam and mathematical function allows a higher density mechanic and gives the player more strategies to create his own solution to the problem.

For a satellite to be used, its icon must be dragged from the inventory area to an asteroid. Only one satellite can be incorporated into each asteroid, so there is only one light beam from such satellite interacting with the other objects on the screen. The number of satellites available in the inventory does not need to be the same of asteroids in the level. An amount of satellites smaller than asteroids challenges the player to select the most suitable satellites and light beams configurations.

An important detail is the ability of satellites to store modules. Modules are another type of item, giving the player access to certain constants of the mathematical formula as it will be described in subsection III-B3. Each satellite has two or three spots for modules that add one more layer of density to the gameplay. If you have more modules, it means more coefficients in the mathematical functions that can be manipulated by the player. Moreover, complex graphs and formulas can be designed as part of the solutions.

2) Crystals: This type of item has an important role in the universe of the game. The discovery of these alien crystals allows building the new defense technologies. In the gameplay, the crystals are responsible for defining which mathematical function will design the light beam placed on the satellites. For instance, the ”Lihne Emerald” generates the light beam as a green straight line from the mathematical formula $y = ax+b$, while the ”Pharabulum Ruby” generates the light beam as a red parabola from mathematical formula $y = ax^2 + bx + c$.

The concept of defining an item to represent the mathematical function in each asteroid, instead of typing directly such function in a text box, aims to keep the coherence and consistency of the game mechanics. Also, by defining previously the type of functions, it becomes possible to create a sequence of levels that shows, one by one, the particular behavior of each mathematical function. The idea is to make the learning process of mechanics and educational content more natural and smooth. The concepts related to the design of the levels and the establishment of their sequences are discussed in subsection III-D.

Only one crystal can be present in one asteroid at a time and the asteroid already has to contain a satellite. To do this, the player must drag the crystal icon present in the inventory to the desired asteroid. It is not necessary that only one type of crystal becomes available in the level inventory. In more advanced levels, it is common to have more than one different type of crystal. This strategy also increases the variability of possible puzzles to be handled during the game, and allows the same challenge to be completed using different types of crystals or even combinations of them. Like satellites, crystals also influence the type of modules in an asteroid. The crystal defines the mathematical function that will be addressed in an asteroid, thus, it is natural that only the coefficients present in this function can be accessed and manipulated by the player.

3) Modules: A module is designed as a device that can be attached to a satellite and connected to a crystal. This device allows the player to control one of the coefficients of the mathematical function, defining the light beam shape. Different modules executes specific changes in the mathematical formulas. The player will have access to one of the coefficients of a specific function that will change within a predefined interval. For instance, in the ”Motor”-type modules, if a Lihne Emerald is attached to an asteroid, the light beam generated has the shape of a straight line. In this case, the player can change the angular coefficient of the line. The ”Hybrid Motor” module allows the player to choose positive values for the coefficient, ”Mirrored Hybrid Motor” allows negative values, and the special golden module ”Hybrid Motor EX” allows setting both negative and positive values. The use of these Motor-type modules can be seen in Figure 4, in which they are used to achieve an optimal solution, with only four satellites, even if there are six asteroids.

Modules designed for a certain type of crystal can be attached only to an asteroid having this crystal, except for the ”Antenna”-type modules. This module allow the addition of an independent constant to any mathematical formula by vertically shifting the graph that rules the light beam. The ”Transmitter Antenna +” allows positive values to be added, ”Transmitter Antenna -” allows negative values and the special golden module ”Transmitter Antenna EX” allows both positive and negative values.

Fig. 4. The optimal solution for the level 2 of the beta version of ”A Plot from the Stars”. Only four Attack Satellites and four Lihne Emeralds can be used to complete the level thanks to the Motor-type modules.

An asteroid can contain two or three modules attached on it, depending on the capacity determined by the satellite. The only rule is that two modules of the same type cannot be present in the same asteroid. Thus, two modules that act on the same constant present in the same mathematical formula are not allowed. At more basic levels, the student is introduced to specific behaviors where only one type of module is use. This will allow him/her to understand the changes in graph behavior as a result of several changes in the coefficients.

To attach a module to an asteroid, the player must drag its icon from the inventory to the desired asteroid where a satellite and a compatible crystal are already present. If a module of the same type is already present, the game replaces it with the newly attached module. If there is not enough space on the satellite for a new module, the player must click on one of the modules already present to replace it.

C. Medals and Extra Objectives

In addition to the main objective of the player - to destroy all the enemy ships present on the screen -, some levels also present extra goals to the player. They consist of restrictions regarding the use of the items from the inventory to find a solution to the puzzle. For instance, restrictions to use only a small number of satellites, a specific crystal or module type. It is also possible to impose or forbid the use of a specific item. A level can have up to two extra goals that can be seen by the player at any time in the level.

The player receives a medal when regular or extra goals are fulfilled. Medals work as a record of the player’s performance and they are cumulative, based on the type of levels satisfactorily completed. A level can lead the player to get one (without extra goals) to three medals (with two extra goals). If the player fails to complete the main goal on one level, she/he will not be able to obtain medals for extra goals, even if she/he meets the requirements for that goal.

After to reach a predefined number of medals, the player ”rises of a patent”. She/he gains access to a new set of levels

where new satellites, crystals and modules will be introduced and used to the puzzles. The amount needed to reach a new set of levels is never equal to the number of levels present in the previous set, i.e., if the player only completes the main goal at all levels of a set, getting only one medal per level, the access to a new set of levels is not granted.

In this way, the medal system plays a double role during the player's engagement process. First, the medals show the evolution of a player, rewarding the player when he/she reaches more complex and interesting solutions instead of simpler ones. Second, the concept of "set of levels" leads medals to play a key role. It will encourage the player to try out new strategies and solutions to problems already known by her/him from previous levels, since the only way to achieve a new set of levels is completing some extra goals.

D. Level Design Guidelines

The level design plays an important part in the gameplay of A Plot from the Stars. They are built to support an important concept related to the player's experience: ensuring that the player has all the information necessary to solve the puzzles. The strategy we use to achieve this goal is to focus on the learning curve of the player. This is done not only about the educational content present in the game, but also taking into account the game interface itself. The first levels present in the first set of missions has the role of showing the player how to perform simple actions, such as setting up an asteroid or checking the medals present on the level, as shown in the Figure 5. They also introduce the player through the concepts of satellite, crystal and module.

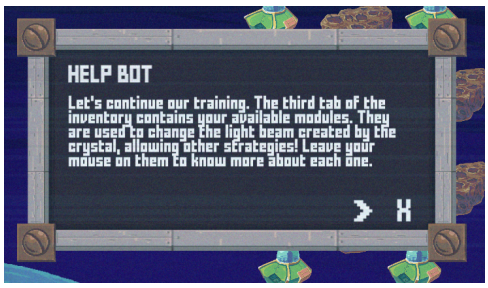


Fig. 5. The beta version of the tutorial system of "A Plot from the Stars". The fictional character "Help Bot" is the player guide to learn about the game interface and mechanics.

In addition, presenting new features of the graphs should also be done gradually from specific levels within a controlled environment. To improve and facilitate the assimilation about how a new type of module can handle a graph, a specific level is available where all its elements are basic and already known by the player. The only concern of the player will be to understand the new concepts aiming to use it at more advanced levels, step by step.

The way that objects in a level are disposed reflects how players will try to identify patterns in the level, when setting a laser beam. For instance, if two enemy ships are very close in a straight line and an asteroid is near them, it is likely that the players understand that a line graph should be placed there. The decision about when using this type of pattern and when subverting it, while game designers, lead us to generate levels that encourage and surprise the player. The levels must guide the player to a creative thinking, which is essential to teach different perspectives from the educational content addressed.

E. Education and Gameplay Together

The main guideline for all the gameplay mechanics presented in this section is to reach a balance between the fun of a conventional digital game and the teaching effectiveness of an educational tool. As presented in section II, it is common the lack of integration between the scope of the game and the scope of education in the educational game context. In "A Plot from the Stars", this balance is the focus of the whole gameplay. The content of mathematical functions does not stay with the game mechanics, but they have been effectively transformed into the mechanics themselves. Crystals and modules are, as presented in subsections III-B2 and III-B3, the very implementation of mathematical functions made in a ludic way with logical challenges through levels. Figure 6 shows the constant change panel, one example of the union between gameplay and content. **The proposed game aims to present itself as an interesting tool for teachers who can make the transition between the playful and the traditional environments. This can be done by teachers that will define more effectively parallels between the content presented in the game with the related ones in the books.**

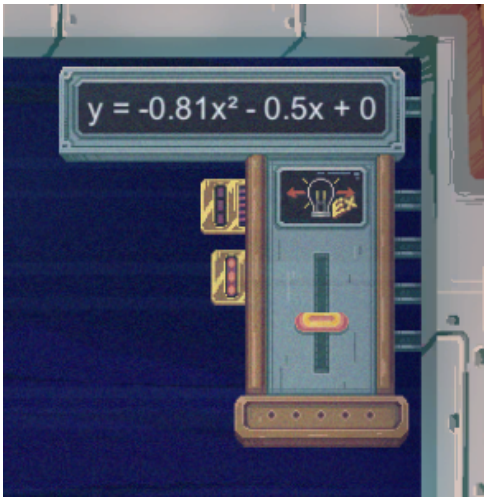


Fig. 6. The constant change panel. This panel is activated when at least one module is attached to a satellite, allowing the player to change the constants of the mathematical formula that governs that light beam.

The gameplay and the user interface are designed to allow an environment conducive to experimentation. The player has to test the different modules and see how they act in the light beams to understand how the constants influence the behavior of the functions. The goal is not to give the player, separately, the abilities to learn to play the game and to learn the educational content. Both concepts are self-contained and

inseparable in our proposal, so we expect that "learn to play" and "learn the content" have almost an equivalence. If the player understands how light beams work, it is intended that he also understands how mathematical function graphs work with the gameplay proposed here.

IV. Results and Discussion

This section presents preliminary exploratory results obtained from an experiment made with a beta version of the game.

A. Game Beta Version

The beta version of "A Plot from the Stars" has 12 levels, where the first 6 levels are dedicated to teaching the game and presenting the items already discussed in subsection III-B. The other 6 levels have puzzles where such concepts are applied. The inventory at these levels is also reduced in comparison to the number of items expected to be in the final version of the game. The Attack Satellite is the only one available and the levels focus on the use of straight lines and parabolas. Therefore, only the crystals and modules related to them are now available. Other mathematical functions such as sine and cosine, among others, will be available in the final version. The beta version is on the Web, https://toschi.itch.io/plotfromthestars, for Linux and Windows operating systems.

B. Research Form

The form submitted to survey participants is available online at https://goo.gl/forms/zc2QRqcVphh0J1pc2. There are questions related to demographic profile, pre-test session, Intrinsic Motivation Inventory (IMI) [16], and a post-test session. The demographic profile evaluates some characteristics of the participants by collecting data such as age, geographic location, sex, educational level and preferencess in the area of games and mathematics. The pre-test and post-test sessions consist of questions about basic math functions, all of them focused on the relationship between the function formula and its graphs. The idea is to measure the learning of participants during their test with the game. The use of the IMI questionnaire aims to evaluate variables related to the participant's experience with the activity of playing.

C. IMI Reliability Analysis

Prior to performing the statistical analysis related to the student's motivation, a factorial analysis and reliability test were conducted in the IMI survey to validate it as an instrument to measure the student motivation. From the collected data of 50 participants, we removed a careless responder who answers the survey with a number of repeated sequential value greater than the half of items, and the reliability analysis was conducted with 49 participants. The participants are a sample of a general audience, ranging in age from 12 to 34, with more people between ages of 17 and 20 (more than 50% of the total). 72% of the participants are men, while 28% are women. A total of 71% of the participants are undergraduate students, while 21% are students from high and elementary school and 8% are graduate students.

Table I shows the result of the exploratory factorial analysis. According to the cumulative variances (> 0.70) and proportion explained (25% for the first component) returned in the factor analysis, our version of IMI questionnaire is consistent with the psychometric validation of original IMI questionnaire [16], [17]. Table II shows the result of the reliability test in the IMI survey. The Cronbach's alpha (α) found for this study has the relevant consistency of 0.90s for all factors.

TABLE I
Explanatory Factor Analysis in the Adapted Portuguese Version of IMI Questionnaire

Scale and Items	F3	F2	F1	F4
Perceived Competence				
I was quite skilled	0.929	0.202	0.044	−0.241
Satisfied with performance	0.836	0.026	0.194	−0.249
I'm good at this activity	0.808	0.006	0.321	−0.094
I could not do it well	−0.791	0.182	−0.180	0.029
I feel competent	0.687	0.321	0.159	−0.280
Good compared to others	0.675	0.394	−0.157	−0.109
Value/Usefulness				
Beneficial activity	0.138	0.940	0.102	0.027
Use for learning	0.019	0.840	0.135	−0.042
Some value to me	0.238	0.802	0.281	−0.009
Activity is important	−0.086	0.798	0.339	0.029
Helps to learn functions	0.129	0.781	0.222	0.024
Interest/Enjoyment				
Activity was fun	0.201	0.207	0.876	−0.020
Interesting activity	0.023	0.367	0.755	−0.071
I really enjoyed doing	0.109	0.369	0.778	−0.328
Very nice activity	0.417	0.207	0.707	−0.269
I reflected about how much I liked	0.058	0.234	0.524	−0.263
Relaxed activity	0.130	0.025	0.704	−0.174
Perceived Choice				
I had no choice	−0.187	−0.012	−0.088	0.908
I felt obliged	−0.167	0.000	−0.177	0.912
I had no choice	−0.245	−0.047	−0.148	0.806
I did it because I had to	−0.123	0.125	−0.370	0.809
Cumulative Variance	0.203	0.405	0.591	0.756
Proportion Explained	0.268	0.267	0.246	0.219

TABLE II
Result of Reliability Analysis for the Adapted Portuguese Version of IMI Questionnaire

	Total
Intrinsic Motivation	0.920
Perceived Competence	0.922
Value/Usefulness	0.935
Interest/Enjoyment	0.910
Perceived Choice	0.942

D. Data Analysis and Discussion

The objective of this analysis is to investigate if there are significant differences in the learning gain, motivation and time

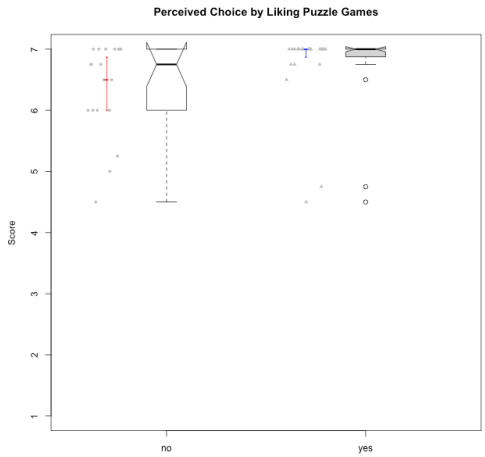


Fig. 7. Perceived Choice IMI scale data by the group of participants that likes puzzle games

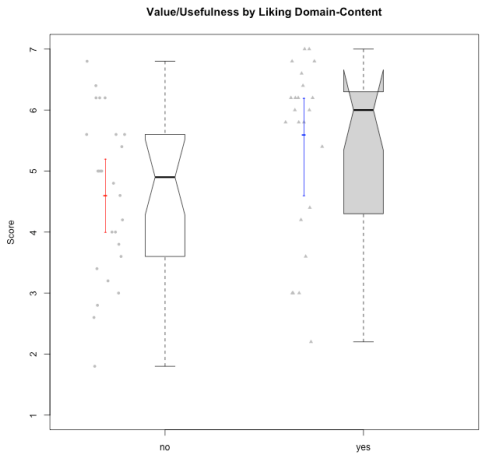


Fig. 8. Value/Usefulness IMI scale data by the group of participants that already likes to know about mathematical functions

spent in the game for the different types of audience classified through the demographic profile. The statistical methods Kruskal [18] and Wilcoxon–Mann–Whitney [19] were used to look for significant differences. The choice for non-parametric methods was due to the reduced sample size and lack of normality in the data collected. Among all the results of the analysis, some interesting aspects could be observed from the IMI questionnaire and demographic profile data results as reported next.

1) Perceived Choice to People That Likes Puzzle Games: By means of a Kruskal-Wallis test, it was possible to observe that there was a statistically significant difference between the value of the Perceived Choice scale for the group of participants who enjoys puzzle games (p = 0.016). The Wilcoxon Signed-Ranks Test indicated that, for this scale, the median ranks for those who enjoy puzzle games (M.Ranks = 25.24) were significantly higher than the median ranks for participants who did not like this genre (M.Ranks = 16.97). This rejects a null hypothesis that "there is no difference in the Perceived Choice scale for those who prefer puzzle games". This allows us to infer that a group preferring puzzle games reveals the tendency to enjoy playing our game. The Figure 7 shows this relationship. Thus, the game being developed tends to attract people who are already familiar with this genre of game.

2) Value/Usefulness and Intrinsic Motivation to People that Like the Domain Content: Another Kruskal-Wallis test reports a statistically significant difference in the "Value/Usefulness" scale of the IMI. This happens for the group of participants that likes the content covered by the game domain: mathematic functions (p = 0.034). A Wilcoxon Signed-Ranks Test indicates that the median ranks (M.Ranks = 29.59) were significantly higher for those who enjoy the content covered. The value is higher than median ranks for participants who dislike mathematical functions (M.Ranks = 20,94). Therefore, it is possible to reject the null hypothesis that "there is no difference in the Value/Usefulness scale for those who have appreciation or not in the field of content display by the game". Thus, for the group that already likes to learn and know about mathematical functions, we can infer a tendency to give more value and importance for the gaming experience. The Figure 8 represents a relationship between these two concepts. This result is interesting to validate the use of the game in a real educational context.

The third Kruskal-Wallis test reports that there is also a significant difference between the value of the "Intrinsic Motivation" scale of the IMI, and the group of participants that like the content covered by the game domain (p = 0.041). A new Wilcoxon Signed-Ranks Test indicated that for this other scale, the median ranks for those who enjoy the content (M.Ranks = 29.43) were also significantly higher than the median ranks for participants who did not like mathematical functions so much (M.Ranks = 21.08). This rejects the null hypothesis that there is also "no difference in the scale of Intrinsic Motivation for those who have appreciation or not, for the domain of the content shown in the game". Thus, we can infer that there is a tendency in players, which already like to learn and to know about mathematical functions, to become more motivated by taking part in the game activity. Figure 9 shows the relationship between these two concepts. This new result, together with those already reported, states again the potential of the gameplay to motivate and catch the players attention. This mainly happens for those who are already interested in the content addressed, increasing their chances of improvement during the learning process.

V. Conclusion

This paper featured an educational game project for mathematics learning, "A Plot from the Stars". We presented its gameplay concepts and preliminary results from an evaluation done by a group of 49 students/players over the beta version of the game. The results indicate that the intrinsic motivation increases when those participants have some knowledge about puzzle games and/or math. This finding is important since it gives information that helps to tailor the game mechanics and features to meet the needs of this type of student. Also,

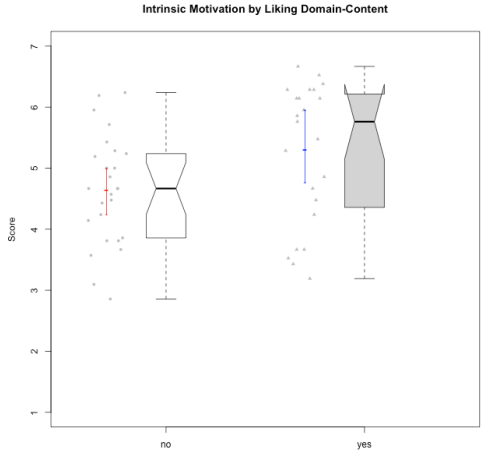


Fig. 9. Intrinsic Motivation IMI scale data by the group of participants that already likes to know about mathematical functions

it can help us to develop new versions of the game aiming to motivate those people who are not interested in math or puzzles. As future work, the gameplay mechanics are expected to be improved as well as more content for the game will be produced. This includes new enemy ships, new items and new mathematical functions such as the sine, cosine and the exponential functions. In addition, new experiments will be conducted within educational environments for the game validation.

REFERENCES

[1] S. L. Kent, *The ultimate history of video games: from Pong to Pokemon — the story behind the craze that touched our lives and changed the world*, 1st ed. Three Rivers Press, 2001.

[2] T. Nath, "Investing in video games: This industry pulls in more revenue than movies, music," 2016. [Online]. Available: https://www.nasdaq.com/article/investing-in-video-games-this-industry-pulls-in-more-revenue-than-movies-music-cm634585

[3] E. S. Association, "2017 essential facts about the computer and video game industry," 2017. [Online]. Available: http://www.theesa.com/wp-content/uploads/2017/09/EF2017_Design_FinalDigital.pdf

[4] A. Insight, "The 2012-2017 worldwide game-based learning and simulation-based markets," *URL: ambientinsight. com/Resources/Documents/AmbientInsight_SeriousPlay2013_WW_GameBase dLearning_Market. pdf*, 2013.

[5] J. Haas, "A history of the unity game engine," *Worcester, UK: Worcester Polytechnic Institute*, 2014.

[6] T. Donovan and R. Garriott, *Replay: The history of video games*. Yellow Ant Lewes, 2010.

[7] C. Shuler, "What in the world happened to carmen sandiego?" in *The edutainment era: Debunking myths and sharing lessons learned. New York, NY: The Joan Ganz Cooney Center at Sesame Workshop*, 2012.

[8] A. Bruckman, "Can educational be fun," in *Game developers conference*, vol. 99, 1999, pp. 75–79.

[9] J. Bidarra, M. Figueiredo, S. Valadas, and C. Vilhena, "O game-book como modelo pedagógico: Investigação e desenvolvimento de um protótipo para ipad," *Aprender na era digital: Jogos e Mobile-Learning*, pp. 83–109, 2012.

[10] T. Pessoa and V. Rios, "A viagem dos pamundo-explorando o potencial educativo em um jogo para ipad," *SBC - Proceedings of SBGames 2011*, 2011.

[11] S. Henderson and J. Yeow, "ipad in education: A case study of ipad adoption and use in a primary school," in *System science (hicss), 2012 45th hawaii international conference on*. IEEE, 2012, pp. 78–87.

[12] C. Hancock and S. Osterweil, "Zoombinis and the art of mathematical play," *Hands on!*, 1996.

[13] C. Fernandez-Vara and S. Osterwil, "The key to adventure game design: Insight and sense-making," *Proceedings of Meaningful Play 2010*, 2010.

[14] B. M. McLaren, D. Adams, R. E. Mayer, and J. Forlizzi, "A computer-based game that promotes mathematics learning more than a conventional approach," *International Journal of Game-Based Learning (IJGBL)*, vol. 7, pp. 36–56, 2016.

[15] J. Groff, J. Clarke-Midura, V. E. Owen, L. Rosenheck, and M. Beall, "Better learning in games: A balanced design lens for a new generation of learning games," *Learning Games Network, MIT Education Arcade*, 2015. [Online]. Available: https://education.mit.edu/wp-content/uploads/2015/07/BalancedDesignGuide2015.pdf

[16] E. McAuley, T. Duncan, and V. V. Tammen, "Psychometric properties of the intrinsic motivation inventory in a competitive sport setting: A confirmatory factor analysis," *Research quarterly for exercise and sport*, vol. 60, no. 1, pp. 48–58, 1989.

[17] V. Monteiro, L. Mata, and F. Peixoto, "Intrinsic motivation inventory: psychometric properties in the context of first language and mathematics learning," *Psicologia: Reflexão e Crítica*, vol. 28, no. 3, pp. 434–443, 2015.

[18] W. H. Kruskal and W. A. Wallis, "Use of ranks in one-criterion variance analysis," *Journal of the American statistical Association*, vol. 47, no. 260, pp. 583–621, 1952.

[19] N. Feltovich, "Nonparametric tests of differences in medians: comparison of the wilcoxon–mann–whitney and robust rank-order tests," *Experimental Economics*, vol. 6, no. 3, pp. 273–297, 2003.

Scenarios for Educational and Game Activities using Internet of Things Data

Chrysanthi Tziortzioti
Hellenic Open University
Patras, Greece
tziortzio@gmail.com

Irene Mavrommati
Hellenic Open University
Patras, Greece
mavrommati@eap.gr

Georgios Mylonas
Computer Technology Institute & Press
Patras, Greece
mylonasg@cti.gr

Andrea Vitaletti
Dep. of Computer, Control and Manag. Eng.
Sapienza University of Rome
Rome, Italy
vitaletti@diag.uniroma1.it

Ioannis Chatzigiannakis
Dep. of Computer, Control and Manag. Eng.
Sapienza University of Rome
Rome, Italy
ichatz@diag.uniroma1.it

***Abstract*—Raising awareness among young people and changing their behavior and habits concerning energy usage and the environment is key to achieving a sustainable planet. The goal to address the global climate problem requires informing the population on their roles in mitigation actions and adaptation of sustainable behaviors. Addressing climate change and achieve ambitious energy and climate targets requires a change in citizen behavior and consumption practices. IoT sensing and related scenario and practices, which address school children via discovery, gamification, and educational activities, are examined in this paper. Use of seawater sensors in STEM education, that has not previously been addressed, is included in these educational scenaria.**

***Index Terms*—IoT, STEM, educational scenarios, behavioral change**

I. Introduction

Wireless Sensor Networks have seen a tremendous development, leading to the realization of the Internet of Things (IoT). Today, there is a large variety of hardware and software to choose from that is easy to set up and use in an increasing number of real-world applications [1]. One such application is education: the deployment of a variety of sensors (e.g., for monitoring electricity consumption, environmental conditions, be them indoor or outdoor, etc.) across school buildings or across different natural water reserves, during different seasons, can produce real-world data to be directly used in STEM educational activities.

This is a crucial period for the future of our planet as it becomes evident that human activities inflict irreversible damage on the environment and on critical resources. One approach for addressing the climate change problem is through the development and transfer of green technologies. Emphasis is given to Environmental awareness via STEM education. This is achieved by sensor experiments, utilizing off the shelf IoT sensors, and educational activities planned specifically for school children, as part of their Science class. Gamification elements are inseparable in such approaches: discovery and adventure are intemperate elements in childs play that leads them through knowledge. Gamification via the internet and social activity mechanisms, on the other hand, is multiplying the impact of the children engagement.

This paper deals with energy and environmental awareness as a part of STEM school educational activities. This is handled in two ways: a) by addressing energy footprint and energy consumption, via individual and group class activities, by using IoT sensors and gamification elements using real sensor data from familiar environments and recording changes in behaviour that affect directly energy consumption. And b) To raise environmental awareness of the systemic nature of changes, (affecting the sustainability of ecosystems, climate, etc) via a quest for inquiry and knowledge using data from sea water sensors. The latter has not been previously addressed due to the difficulties imposed by the nature of such IoT infrastructure that can be used for getting data in sea water.

Reinforcing the educational community on educating the new generations will create a multiplier on the overall energy reductions: promoting sustainable behaviours at school will also reflect behaviours at home. Several studies document the ability of students to influence choices made by their families related to environmental issues [2]. The research interviews conducted in [3] made clear that energy conservation insights learned in school can be applied at home by students and their families. Since about 27% of EU households include at least one child under the age of 18 [4], targeted efforts of reaching families of children and young people will scale further to reach a large portion of the EU population and multiply the benefits towards sustainability of the planet.

The rest of the paper is structured as follows. In Sec. II relevant literature is presented and in Sec. III a platform for using data collected from IoT deployments at schools in STEM is presented in details. In Sec. IV a set of educational actions is presented that uses the data collected from the IoT infrastructure to focus on promoting behaviour change. In Sec. V a new set of educational scenarios are proposed for examining the aquatic sector and how to educate students on

978-1-5386-4359-4/18/$31.00 ©2018 IEEE

sustainable behaviours. The paper concludes in Sec. VI where future research directions are also provided.

II. Related work

The approach of promoting sustainable behavioural change through activities targeting the educational sector falls within the scope of several research projects. [5] focuses on 4 public university sites with pilot hardware installations, combined with software aimed either at desktop users or mobile users, for promoting energy consumption awareness and engagement. Focused on school buildings are the Veryschool [6] and Zemeds [7] projects, producing recommendation and optimization software components, or methodologies and tools. [8] produced several guidelines and results regarding good energy saving practices in an educational setting.

The procedures proposed (see sections IV and V) include the users monitoring data and drawing subsequent conclusions - as a part of games or school education assignments- as a first step towards raising awareness. The concept of users in the loop of monitoring is central in the area of participatory sensing [9] in which personal mobile phones of users are used to collect relevant data for a number of applications such as urban planning, public health, cultural identity and creative expression, and natural resource management. This approach has been employed by the Cornell Laboratory of Ornithology [10] in a science education project on bird biology, while in [11] the authors describe trials for air quality, water quality and plant disease monitoring. Similarly to our context, [12] presents a solution combining a deployed and participatory sensing system for environmental monitoring.

Therefore facilitating the development of diverse application scenaria and supporting the different requirements in terms of data interpretation and analytics is a crucial aspect. In [13] look into people-centric applications for facilitating the educational sector towards improving the energy efficiency of school buildings. In a broader context, in [14] people-centric scenaria are examined at a smart city level.

Other related approaches are reporting on IoT enabled gamification, targeting reduced energy consumption in public buildings [15] and [16].

An integration of an IoT data management platform and a serious game, whereby users compete in energy-related actions is reported in: [17]. In the past several approaches have been proposed in order to address the potentially huge number of sensor data arriving from the IoT domain, each one of them applied in different parts of the network architecture [18]–[21]. In the GAIA project approach, a platform is used for sensor reading related gamification activities, referred to as ”the GAIA challenge”, which can be seen in [22].

In [23] the authors discuss the value of participating to project like these for students, concluding that “Students are gaining deep domain-specific knowledge through their citizen science campaign, as well as broad general STEM knowledge through data-collection best practices, data analysis, scientific methods, and other areas specific to their project”

III. IoT and Real-world Data in STEM

One approach for addressing the climate change problem is through the development and transfer of green technologies. In the context of reducing the energy spent in residential buildings, new technologies have been introduced that improve the energy efficiency of buildings. In fact, till now the dominant approach was to use energy-efficient infrastructure and materials to reduce the energy consumption of buildings. Unfortunately, the rates of construction of new buildings as well as the rates of the renovation of existing buildings are both generally very low [24] to expect a significant effect on the total amount of energy spent in our everyday life at a global level. Similarly, the approach for reducing the energy consumption in transportation focuses on improving the energy efficiency of motor engines. Also here, given the rate of change of existing fleets with energy efficient one, it is very challenging to save energy in this sector through this approach [25].

An alternative approach, that has recently received emphasis, is the promotion of energy consumption awareness, sustainability and behavioural change on people. The main concept is that to address the global climate problem requires informing the population about their roles in mitigation actions and adaptation of sustainable behaviours. In other words, addressing climate change and achieve ambitious energy and climate targets requires a change in citizens' behaviour and consumption practices [26]. Reports indicate that citizens making efficient use of energy in their everyday life can lead to large energy and financial savings and potentially to a substantially positive environmental impact [26].

A key challenge for achieving sustainability and transforming people's behaviour towards energy consumption is the need to educate them on such issues. An interesting starting point is the educational sector. Raising awareness among young people and changing their behaviour and habits concerning energy usage is key to achieving sustained energy reductions. At EU level, people aged under 30 represent about a third of the total population [27]. Thus, by targeting this group of citizens we affect a large part of the EU population. Additionally, young people are very sensitive to the protection of the environment so raising awareness among children is much easier than other groups of citizens (e.g., all attempts made to achieve behavioural change and establish new environment-friendly habits to children regarding recycling have had very high success rates).

The GAIA platform [28] is among the very few IoT systems that have focused on the educational community. A real-world IoT deployment is spread in 3 countries (Greece, Italy, Sweden), monitoring in real-time 18 school buildings in terms of electricity consumption and indoor and outdoor environmental conditions. The data collected is used as part of series of education scenarios whose goal is to educate, influence and attempt to transform the behaviour of elementary school students through a series of trials conducted in the educational environment and in homes. Feedback mechanisms

notify the students on current energy consumption at school and in this way assist towards raising awareness regarding environmental effects of energy spending and promote energy literacy by educating the users.

GAIA is based on the principle that continuously monitors the progress of students positively contributes towards reducing the energy consumption and successful behaviour change. Since the IoT deployment is multi-site and multi-country can motivate, for example, to identify energy consumption patterns in different countries and across different climate zones. This can be used to make comparisons or competitions; for instance, students of school A compete with students of school B inefficiency. This could also help understanding cultural differences with respect to energy efficiency awareness and sustainability.

The deployed devices provide 880 sensing points organized in four main categories: (1) classroom environmental comfort sensors (devices within classrooms); (2) atmospheric sensors (devices positioned outdoors); (3) weather stations (devices positioned on rooftops); and (4) power consumption meters (devices attached to the main breakout box of the buildings, measuring energy consumption). Given the diverse building characteristics and usage requirements, the deployments vary from school to school (e.g., in number of sensors, hardware manufacturer, networking technology, communication protocols for delivering sensor data, etc.). The IoT devices used are either open-design IoT nodes (based on the Arduino popular electronics prototyping platform, see [29]) or off-the-shelf products acquired from IoT device manufacturers.

The platform also incorporates participatory sensing technologies for semi-automatic periodical collection of energy usage to acquire information in buildings where no IoT sensing elements are available, e.g., utilizing web/smartphone/social networking applications for acquiring information on room occupancy, usage of conditioning or special machinery, opening of windows, etc. The goal of GAIA is to include the users in the loop of monitoring the energy consumption in the buildings they use daily, thus making the first steps towards raising awareness, connecting the educational activities carried out at schools with their activities at their home environment and also engaging the parents and relatives at home. The teacher can initiate participatory sensing sessions during the courses so that students can use phones and tablets to gather data in real time and then review them in class (for more information see Sec. IV).

The integrated sources of input are utilized to continuously provide direct feedback, custom-tailored to each particular learner/audience (i.e., kindergarten, school, university, parents). Direct feedback is provided via real-time displays (RTDs) installed at central locations in the buildings, published on school websites, posted to social media, and also displayed on the users' smartphones and tablets. Direct feedback mechanisms are developed to address the immateriality of energy [30] and make it a visible entity by connecting it to the daily activities of students. Visual analytics are combined with recent advances in IoT sensing and pervasive computing technologies to provide an interactive environment that *stimulates behavioural change on a frequent basis*. The energy consumption topic is included in the pedagogical activities of the schools incorporating educational aspects to promote energy literacy, convey information regarding historical data and comparative information with other buildings of similar characteristics (for more information see Sec. IV).

A series of social-networking applications are provided to set community-based incentives for pro-environmental behavioural change and promote collective consuming of resources. These applications utilize the already established relationship between users of the same school/department to provide *community-based initiatives* to reduce their overall environmental footprint and increase environment-friendly activities. A series of *game-based competitions* further engage the students in learning how to improve the energy efficiency, and to encourage them to actually follow the learned practices. Research suggests that competitions can be effective in promoting environmentally responsible behaviour [31]. Historical data collected from the IoT infrastructure allows students to compete with each other on periodic intervals (e.g., per week/month/season) to further motivate eco-friendly behaviours. A combination of *direct competition* among other groups of similar size, climate zone, socio-economic characteristics and past years (e.g., class 2016 vs recordings from class 2015) and *indirect competition* against each group's own performance is followed. These competitions *encourage spreading the word to larger groups*, allowing related persons, such as parents, friends, or neighbours, to participate and also *appeal to positive emotions*, such as hope and enjoyment, as ways to changing individuals behaviours.

Bringing IoT into the sea. Most of the works in the IoT domain focus on terrestrial applications. Even when offshore infrastructures or vessels are considered, IoT devices are mostly deployed in "dry" surfaces and only some specific transducers are actually deployed into the water. The underwater environment is hostile, and consequently, underwater IoT devices are very expensive. If you only consider a reliable water-proof housing for shallow water, it costs at least 2 or 3 order of magnitude more than terrestrial solutions and much more if you consider deep water scenarios. Underwater operations are complex and challenging. As an example, the fast growth of algae or microorganisms can suddenly affect the quality of sensors readings that have to be often cleaned. Underwater communications are still extremely difficult and energy-hungry; RF propagates only a few centimetres and only acoustic or optical communications can be used for longer distances. The energy cost of underwater communications strongly limits the device lifetime, that is usually in the order of few months at best and requires frequent replacements of the batteries, an annoying, time-consuming and difficult task. Finally, communication standards are emerging only in the last years. Due to these reasons, the availability of underwater IoT data is still very limited. One of the few attempts to provide a federation of underwater testbeds for the Internet

of Underwater Things is the EU project SUNRISE [32]. While SUNRISE clearly showed us the potential of exploring underwater data, it was not originally conceived for STEM educational activities, and both the complexity of the tools and the costs of the equipment are not yet suitable to be operated by students. Despite these difficulties, there are already some efforts for more affordable tools for underwater investigations [33], [34] and is, however, possible to design significant STEM activities (see section V) that focus on shallow water and/or surface sampling that significantly lower the above-discussed difficulties. Indeed, the focus on the shallow water and/or the sea surface allow us to a) engage students in participatory sampling (i.e. they are directly involved in the sampling procedure at sea), b) deploy relatively simple networking infrastructures capable to deliver the data acquired by possible underwater traducers employing standard wireless technologies (e.g. Lora, Sigfox or even WiFi). In the latter case, the transducers can be placed underwater and the collected data are delivered by a cable to a wireless device on the surface that makes them available in the cloud.

IV. Energy Efficiency Education

A main objective of environmental sustainability education in terms of raising awareness towards energy efficiency is to make students aware that energy consumption is largely influenced by the sum of individual behaviours (at home, school, etc.) and that behaviour changes and simple interventions in the building (e.g., replacing old lamps with energy-efficient ones) can have a great impact on achieving energy savings. IoT technologies can support these initiatives by mediating people's interaction with the environment in order to provide immediate feedback and actually measures the impact of human actions while automating the implementation of energy savings policy and at the same time maintaining the comfort level perceived by people.

Teachers can use collected data and analytics during class to explain to pupils basic phenomena related to the parameters monitored and organize student projects, where each student monitors specific environmental parameters at their home. In Monitoring school buildings situated in different countries can help, e.g., to identify usage or energy consumption patterns. This, in turn, can be utilized to make comparisons or realize competitions through social networking and game applications (e.g., students of school A compete with students of school B in answering energy awareness questions).

Including the users in the loop of monitoring their daily energy consumption is a first step towards raising awareness. In an educational environment, this step can be further enhanced and capitalized in the framework of educational activities with the support of the IoT infrastructure. The educational activities in each school are based on data produced within the respective buildings, while the effects of changing certain behaviours can be detected and quantified. E.g., teachers can complement existing educational activities on sustainability with simple actions with immediate IoT-enabled feedback, such as turning off the lights in parts of the building and monitoring the drop in consumption or using thermal cameras to discover problematic areas combined with data showing the effect of incomplete building insulation.

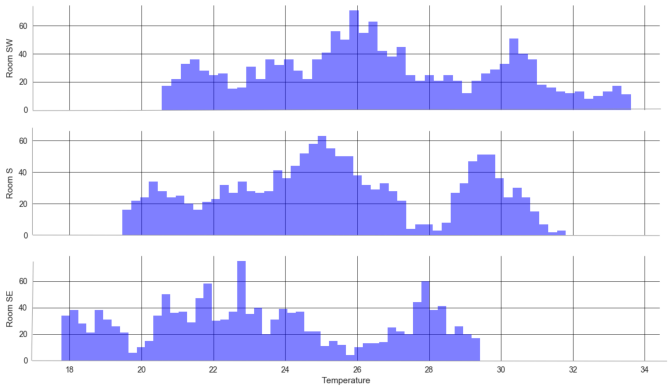


Fig. 1: Indoor temperature histogram for three classrooms during Sep/17 to Oct/17

Scenario 1: The Importance of Building Orientation is the practice of facing a building so as to maximize certain aspects of its surroundings, such as street appeal, to capture a scenic view, for drainage considerations, etc. With rising energy costs, it is becoming increasingly important for builders to orient buildings to capitalize on the Sun's free energy. In this scenario, the students are introduced to basic notions of building orientation and how to take advantage of the sun warmth to increase indoor comfort and reduce energy consumption. IoT sensors that monitor indoor temperature and humidity are used to observe how indoor conditions vary throughout the day. Data collected from the other classrooms of the school are used for comparing the indoor conditions of rooms with a different orientation. The educational scenario provides information on how to reconstruct the surroundings of the building in order to affect the effects of the sun. As example trees are an important factor in passive solar design because they can provide shade during hot summer days. Data collected from classrooms of similar orientation where however there are different trees located on the outside are used to observe how indoor conditions vary.

In Fig. 1 a histogram is provided for the indoor temperature of the three classrooms (facing south, south-west and south-east) examined during a period of 2 months. Lower temperatures are observed in the room facing South-East in contrast to the other two rooms. Examining the classrooms temperature is a measure of understanding the conditions under which students and teachers operate. Hot, stuffy rooms—and cold, drafty ones—reduce attention span and limit productivity.

Scenario 2: Insulation Materials have a critical impact on the indoor conditions of classrooms during the daily educational activities. Evaluating the indoor conditions of a building also requires considering other factors related to the construction materials used, the location of the windows and the heating and ventilation technology used. The GAIA platform includes

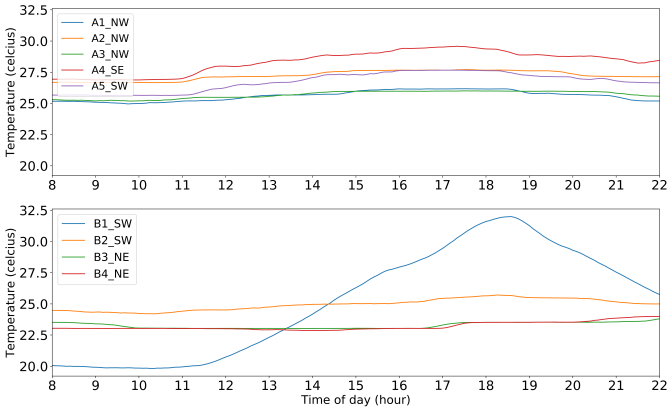


Fig. 2: Classroom temperature during 30/Sep (Saturday)

school buildings located in different climatic zones, constructed in different years ranging from 1950 to 2000, using diverse materials and with different heating and ventilation systems. The education scenario uses the data collected from separate buildings in order to demonstrate the behaviour of temperature and humidity across buildings located in similar climatic zones which however have different construction methodology.

Given the above considerations, the temperature of each room is examined to identify poorly performing classrooms. In Fig. 2 two specific performance issues regarding two schools located in the same city are depicted. The first issue is related to the bottom figure, where room R1 achieves very poor performance with temperature starting at the very low level of 20^oC and increasing up to 32^oC within 8 hours. The second issue is related with the top figure, where the south-west facing classroom (R5) and the south-east facing classroom (R4) have an increase of 2 degrees during the day while all the other rooms are not affected. Even the south-west facing room R2 of the bottom figure does not have such an increase during the day. After contacting the school building managers it was reported that (a) room R1 (bottom school) is located outside the main building, within a prefabricated iso box where insulation is very poor and (b) rooms of top school have no window blinds installed in contrast to the bottom school where window blinds are installed in all rooms. These are just two examples of the results of the analysis conducted. It is expected that such an analysis can provide strong evidence on how to improve the performance of schools.

During spring 2017, a set of preliminary testing was conducted over several weeks to get feedback regarding the educational scenaria that promote energy efficiency and sustainability. A total of 944 students and teachers had the first interaction with the GAIA platform, while we conducted a form-based survey focusing on the gamification component (196 high-school students in Sweden and Italy) and the Educational Lab Kit (132 6th graders in Greece). With respect to the game, 78% of the students found the content interesting (21% extremely, 26% very, 31% moderately) and 89% the activity user-friendly (38% extremely, 29% very, 22% moderately). Regarding the acceptance of the tools from educators, the direct response gathered through workshops has been positive and several schools have provided their own schedules for integrating GAIA tools in classes. Thus, in terms of overall acceptance of both the tools and the infrastructure inside buildings and the schools curricula, the results indicate that the educational scenaria had a quite positive response.

V. Sea Polution Education

Most of the planet's surface is covered by the sea. Specifically, about 79% of the surface of the Earth is covered by water and only 21% of the land. Today, we know the great importance of the sea for life on the entire planet and especially for humans. The sea has a multiple importance as being a "source of life" for Earth. It provides the ability to produce food, minerals and energy, is a key factor for the renewal of the oxygen we breathe, and the means of transporting goods (trade, energy transfer/information). Maritime trade routes have also been cultural bridges, integrating culturally large and disperse geographic areas and allowing the development of cultures. However, in order to achieve better use of the potential of the sea and at the same time to effectively protect it, a detailed study is required [35].

In the educational scenarios proposed, a series of sensors are used to measure physical and chemical marine parameters. As already observed in section III, bringing the IoT into the sea is still very difficult, for this reason, we will focus on surface sampling activities that are more affordable in the context of STEM educational activities.

The steps of the pedagogical activities we follow are awareness, observation, experimentation and action. School students located in Europe's coastal areas use portable equipment to carry out relevant measurements and submit them to a database they have access to. Depending on the teaching needs and priorities, students can collect and analyze the following:

- current values and any fluctuations of them during the observation period of the activity,
- changing values for longer periods of time, e.g. making comparisons between different times of the day, between months, seasons, or years,
- the variance of the phenomena between different areas.

The mathematical and scientific thinking developed in the above process can be exploited in various ways by the tutor, in the context of teaching mathematical and scientific skills, not only in the science courses but also in cross-thematic approaches that combining such observations and analyzes the economic, social and other aspects of our effort for clean seas.

Scenario 1: Observation of Sea Water Temperature is achieved via water temperature sensors positioned at the surface of sea level. Surface water temperature has a natural daily (diurnal) and seasonal variation due to weather conditions and thermal exchanges with the atmosphere. Students use the IoT infrastructure during the school year to observe the temperature of the surface water of the sea, examining measurements

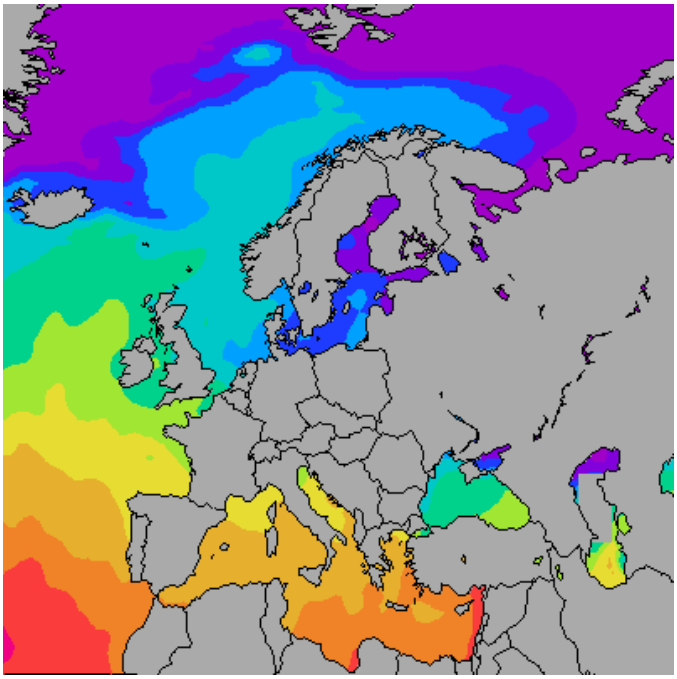


Fig. 3: Surface temperature of the seas of Europe

at different times of the day and at different depths (up to $10m$) and experimentally confirm their theoretical predictions.

Given that GAIA platform is deployed across different countries, *the data collected can accommodate the study of the surface temperature of the sea in relation to latitude*. The surface temperature distribution fully corresponds to the distribution of the solar radiation entering the sea. The global ocean surface temperatures (for water depths up to $5m$) show a bandwidth in terms of latitude. Near the Equator, the waters have high temperatures throughout the year. On the contrary, in the areas near the poles, the temperatures of the surface layers are almost always very low. In the temperate climate zone, the temperature values obtained by the surface water mass, are lower than those of tropical waters and higher than the corresponding polar waters and change significantly during the year [25].

Students in the Mediterranean coastal regions will record the higher surface temperature on the same day of the year than pupils in the Baltic or Atlantic coastal regions, due to the different latitude and hence to different amounts of solar radiation that the region receives.

The historic records collected from the IoT infrastructure also enable to study the temperature of the sea-surface during the course of a year. Students using the measurements they recorded during the year will be able to explain the seasonal variation of surface temperature. In winter, the waves of the sea are more intense and the surface layer is being mixed while the temperature is low and uniform. In the summer, where the atmospheric temperature is high and the wave intensity is small, water mixing is minimal and the temperature of the surface layer increases strongly due to heat build-up. From March to August, the temperature on the surface of the sea is constantly increasing due to the absorption of heat from the atmosphere. So, in the spring, the sea surface begins to create seasonal thermocline, which has a small thickness and small temperature range. In summer, the surface temperature is constantly increasing due to the high temperatures prevailing in the atmosphere, resulting in the seasonal thermocline becoming thicker and higher in the temperature range. From September to February, the surface layer is constantly losing heat. The temperature of the atmosphere is smaller than the sea, and the intensity of the waves is constantly increasing so that it is fully agitated. So in autumn the thermocline thickness increases, but its width decreases relative to summer. In winter, the decrease of the surface layer continues, with the result that the temperature becomes uniform up to the ceiling of the permanent thermocline.

The annual range of surface temperatures is maximal at the intermediate latitudes, while at the small and large geographical heights the range is minimal. School pupils in medium latitudes, due to the high fluctuations in atmospheric temperature and other atmospheric phenomena, expect to record a higher time variation in the temperature of the surface layer.

The educational scenario can be extended to include measurements in shallow coastal areas, where seasonal variation may be absent from the general rule, particularly if there is a significant effect of brackish waters. The students conclude that local water mass inflows and the resulting mixing of different water types, a phenomenon particularly common in coastal waters, may lead to deviations from the general rules.

Finally, the capability of the IoT infrastructure to *continuously collect data in real-time allows the study of the surface temperature of the sea during the day.* Apart from the seasonal thermocline in the middle latitudes, there is also the daily thermocline (diurnal), which is particularly pronounced in the spring, summer and autumn. Groups of students from schools in different coastal regions of Europe collect daylight surface temperature data, record the values in the database, process the data, and arrive at scientific conclusions about the variation in surface sea temperature.

Scenario 2: Observation of Sea Water Acidity-Alkalinity is achieved by deploying pH sensors[1] across different coastal sites. pH plays a major role in the marine ecosystem because it determines the solubility and chemical form of most of the substances present in it. The reduction or increase in pH is directly related to the photosynthesis and respiration of the marine ecosystem organisms and therefore is related to the productivity of the biomass. On the surface of the sea the pH ranges from 8.0 to 8.3 and depends on the atmospheric pressure of the CO, the temperature and the salinity of the water. Students find that the chemical properties of seawater differ from those of the sweet because of the presence of salts. The less acidic salts contained in seawater (such as carbonates, bicarbonates and borates) reduce the high acid or alkaline composition of any liquid waste. So the toxicity of

[1]Gravity: Analog pH Sensor / Meter Kit For Arduino

the wastewater is high in the freshwater and decreases in the sea. pH measurement is the best way to assess the effects of acid or alkaline waste disposal on the marine ecosystem.

An additional goal of this activity is for students to understand that the critical survival limit for life in lakes and water streams does not depend on the average value of pH (degree of accelerating) over a year but on the lowest value of pH. Such short but dangerous periods with low pH values occur, mainly in the spring during the melting of the ice (acidity shocks). Fluctuations in pH may result in the death of many organisms (e.g., plankton at 6.5 and perch and eel at 6.4 and $6.3 - 6.5$ respectively). If the pH value is below 6.5, the adverse effects on all living organisms begin and below pH 5 all animals and plants die.

Scenario 3: Observation of Sea Water Salinity and Conductivity is achieved by deploying salinity sensors[2] (S) and conductivity sensors[3] (STD) across different sites of the IoT deployment. Students in different coastal regions of Europe record sea salinity values and conclude that the total mass of dissolved salts varies from one sea to another, exceeding $36grams$ in the Mediterranean Sea and falling below 10 grams of salt per kg of water in some areas of the Baltic Sea. Students conclude that surface salinity is greatest at latitudes where annual evaporation is greater than annual rainfall and minimum salinity values are found at latitudes where rainfall is greater than evaporation. River water also affects salinity values, for example, the Baltic Sea is a basin with limited communication with the Atlantic, where large rivers are poured, while evaporation is minimal. On the contrary, the Mediterranean and the Red Sea are two basins where the exhaust is large and the discharge of river water is minimal, resulting in large amounts of salinity. In addition, melting and ice formation plays a role in Polar Regions.

Scenario 4: Observation of Sea Water Turbidity is achieved by deploying turbidity sensor[4] (T) across different sites of the IoT deployment. Students using the turbidity sensor record physical parameter values that determine the ability of sunlight to pass through the water. Turbidity is caused either by natural causes (erosion or decomposition of organisms after their death) or by the colloidal and fine-grained suspended solids contained in sewage and industrial waste and precipitating at the bottom with great difficulty and directly affecting ecosystem species with increased need light for their development. The depth of penetration of light in seawater is critical for primary production (photosynthesis) and depends on the clarity of seawater and the wave of light radiation.

Scenario 5: Observation of Dissolved Oxygen Sensor is achieved by deploying dissolved oxygen sensors[5] (DO) across different sites of the IoT deployment. Wastewater from our houses contains organic substances that can be used as feed by other organisms, particularly microbes. These organisms with oxidative reactions metabolize organic substances by consuming for this process the oxygen dissolved in the water. Because oxygen has relatively little water solubility, it is quickly consumed when there is a high organic load resulting in anaerobic conditions. Concentration less than $7mg/lt$ means oxygen deficiency resulting in the non-survival of fish and other aerobic organisms. Physiological values of DO range above $7mg/lt$. Sensors of turbidity and dissolved oxygen will be used to study the phenomenon of eutrophication on closed shores - gulfs where water circulation is limited, near coastal rural areas or in areas near ports or in areas where sewage flows into the marine environment without being biologically cleaned. Apart from the areas where eutrophication affects the environment, students will also identify areas such as river estuaries, which tend to be naturally eutrophic, because they transport nutrients to the open sea, giving increased productivity and food to fish and other organisms. It is not by chance that important sea fishing grounds are located near estuary areas (North Aegean, Thracian Sea, etc.).

[2]Vernier Salinity Sensor
[3]Vernier Conductivity Probe
[4]Vernier Turbidity Sensor
[5]Vernier Dissolved Oxygen Probe

VI. Conclusions

This paper reports on the use of IoT sensors as a basis for educating children in environmental awareness and STEM. A number of educational scenarios are presented based on sensors readings from school buildings as well as seawater measurements. The paper raises the subject of early awareness of environmental issues based on real-world data and the use of motivating / gamified scenarios.

An education-focused real-world IoT deployment in schools in Europe can help promote sustainable activities. By using this infrastructure and the data it produces, it is easier and more effective to build tools that better reflect the everyday reality in school buildings and provide a more meaningful feedback. The development of educational activities focusing on energy awareness in schools has received very positive feedback from the educational community. We believe that recent technological developments allow us to extend the IoT infrastructure in order to monitor additional environmental parameters apart from energy consumption. Towards this end, we propose the deployment of sensors in the sea to form underwater sensor networks for monitoring the aquatic sectors of our planet. A series of educational scenaria that utilize the collected data to further promote sustainability awareness and behavioural change. This can be achieved via educational activities in schools as well as gamification, facilitated by platforms such as the one realized in the GAIA project.

VII. Acknowledgments

This work has been partially supported by the EU research project “Green Awareness In Action” (GAIA), funded under contract number 696029 and the research project Designing Human-Agent Collectives for Sustainable Future Societies (C26A15TXCF) of Sapienza University of Rome. This document reflects only the authors’ view and the EC and EASME are not responsible for any use that may be made of the information it contains.

REFERENCES

[1] I. Chatzigiannakis, G. Mylonas, and A. Vitaletti, "Urban pervasive applications: Challenges, scenarios and case studies," *Computer Science Review*, vol. 5, no. 1, pp. 103–118, 2011. [Online]. Available: https://doi.org/10.1016/j.cosrev.2010.09.003

[2] C. Schelly, J. E. Cross, W. S. Franzen, P. Hall, and S. Reeve, "How to go green: Creating a conservation culture in a public high school through education, modeling, and communication," *Journal of Environmental Education*, vol. 43, no. 3, pp. 143–161, 2012.

[3] K. Crosby and A. B. Metzger, "Powering down: A toolkit for behavior-based energy conservation in k-12 schools," U.S. Green Building Council (USGBC), Washington DC, USA, Tech. Rep., 2012.

[4] "Household structure in the eu," Statistical Office of the European Union (Eurostat), Tech. Rep., 2010, iSBN: 978-92-79-16760-7.

[5] J. Medina, M. Nina, and Ivaro Oliveira, "Smart campus - building-user interaction towards energy efficiency through ict-based intelligent energy management systems," in *European Project Space on Information and Communication Systems - EPS Barcelona,*, INSTICC. ScitePress, 2014, pp. 11–30.

[6] M. Brogan and A. Galata, "The veryschool project: Valuable energy for a smart school - intelligent ISO 50001 energy management decision making in school buildings," in *Proceedings of the Special Tracks and Workshops at the 11th International Conference on Artificial Intelligence Applications and Innovations (AIAI 2015), Bayonne, France, September 14-17, 2015.*, 2015, pp. 46–58. [Online]. Available: http://ceur-ws.org/Vol-1539/paper5.pdf

[7] N. Gaitani, L. Cases, E. Mastrapostoli, and E. Eliopoulou, "Paving the way to nearly zero energy schools in mediterranean region - zemeds project," *Energy Procedia*, vol. 78, pp. 3348 – 3353, 2015, 6th International Building Physics Conference, IBPC 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1876610215024819

[8] "School of the Future FP7 project," 2016, http://www.school-of-the-future.eu/.

[9] J. Burke, D. Estrin, M. Hansen, A. Parker, N. Ramanathan, S. Reddy, and M. B. Srivastava, "Participatory sensing," in *In: Workshop on World-Sensor-Web (WSW06): Mobile Device Centric Sensor Networks and Applications*, 2006, pp. 117–134.

[10] D. Brossard, B. Lewenstein, and R. Bonney, "Scientific knowledge and attitude change: The impact of a citizen science project," *International Journal of Science Education*, vol. 27, no. 9, pp. 1099–1121, 2005.

[11] V. Kotovirta, T. Toivanen, R. Tergujeff, and M. Huttunen, "Participatory sensing in environmental monitoring – experiences," in *2012 Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, July 2012, pp. 155–162.

[12] W. Sun, Q. Li, and C. K. Tham, "Wireless deployed and participatory sensing system for environmental monitoring," in *2014 Eleventh Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*, June 2014, pp. 158–160.

[13] D. Amaxilatis, O. Akrivopoulos, I. Chatzigiannakis, and C. Tselios, "Enabling stream processing for people-centric iot based on the fog computing paradigm," in *22nd IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2017, Limassol, Cyprus, September 12-15, 2017*, 2017, pp. 1–8. [Online]. Available: https://doi.org/10.1109/ETFA.2017.8247674

[14] V. Gutiérrez, D. Amaxilatis, G. Mylonas, and L. Muñoz, "Empowering citizens toward the co-creation of sustainable cities," *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 668–676, 2018. [Online]. Available: https://doi.org/10.1109/JIOT.2017.2743783

[15] T. G. Papaioannou, D. Kotsopoulos, C. Bardaki, S. Lounis, N. Dimitriou, G. Boultadakis, A. Garbi, and A. Schoofs, "Iot-enabled gamification for energy conservation in public buildings," in *Global Internet of Things Summit, GIoTS 2017, Geneva, Switzerland, June 6-9, 2017*. IEEE, 2017, pp. 1–6.

[16] G. Mylonas, D. Amaxilatis, H. Leligou, T. B. Zahariadis, E. Zacharioudakis, J. Hofstaetter, A. Friedl, F. Paganelli, G. Cuffaro, and J. Lerch, "Addressing behavioral change towards energy efficiency in european educational buildings," in *Global Internet of Things Summit, GIoTS 2017, Geneva, Switzerland, June 6-9, 2017*. IEEE, 2017, pp. 1–6. [Online]. Available: https://doi.org/10.1109/GIOTS.2017.8016258

[17] C. Garcia-Garcia, F. Terroso-Saenz, F. Gonzalez-Burgos, and A. F. Skarmeta, "Integration of serious games and iot data management platforms to motivate behavioural change for energy efficient lifestyles," in *Global Internet of Things Summit, GIoTS 2017, Geneva, Switzerland, June 6-9, 2017*. IEEE, 2017, pp. 1–6. [Online]. Available: https://doi.org/10.1109/GIOTS.2017.8016255

[18] I. Chatzigiannakis, G. Mylonas, and S. E. Nikoletseas, "50 ways to build your application: A survey of middleware and systems for wireless sensor networks," in *Proceedings of 12th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2007, September 25-28, 2007, Patras, Greece*. IEEE, 2007, pp. 466–473. [Online]. Available: https://doi.org/10.1109/EFTA.2007.4416805

[19] I. Chatzigiannakis, G. Mylonas, and S. Nikoletseas, "jWebDust: A java-based generic application environment for wireless sensor networks," in *Distributed Computing in Sensor Systems, First IEEE International Conference, DCOSS 2005, Marina del Rey, CA, USA, June 30 - July 1, 2005, Proceedings*, ser. Lecture Notes in Computer Science, vol. 3560. Springer, 2005, pp. 376–386. [Online]. Available: https://doi.org/10.1007/11502593_29

[20] I. Chatzigiannakis, H. Hasemann, M. Karnstedt, O. Kleine, A. Kröller, M. Leggieri, D. Pfisterer, K. Römer, and C. Truong, "True self-configuration for the iot," in *3rd IEEE International Conference on the Internet of Things IOT*, 2012, pp. 9–15.

[21] I. Chatzigiannakis, A. Kinalis, and S. E. Nikoletseas, "Power conservation schemes for energy efficient data propagation in heterogeneous wireless sensor networks," in *Proceedings 38th Annual Simulation Symposium (ANSS-38 2005), 4-6 April 2005, San Diego, CA, USA*. IEEE Computer Society, 2005, pp. 60–71. [Online]. Available: https://doi.org/10.1109/ANSS.2005.37

[22] "The GAIA Case," 2017, https://iot.ieee.org/newsletter/november-2017/green-awareness-via-iot-infrastructure-educational-labs-and-games-in-schools-the-gaia-case.

[23] S. Heggen, "Participatory sensing: Repurposing a scientific tool for stem education," *interactions*, vol. 20, no. 1, pp. 18–21, Jan. 2013. [Online]. Available: http://doi.acm.org/10.1145/2405716.2405722

[24] "Europe's buildings under the microscope: A country-by-country review of the energy performance of buildings," Buildings Performance Institute Europe (BPIE), Tech. Rep., 2011, iSBN: 9789491143014.

[25] P. Capros, L. Mantzos, V. Papandreou, and N. Tasios, "European energy and transport trends to 2030," Institute of Communication and Computer Systems of the National Technical University of Athens (ICCS-NTUA), E3M-Lab, Greece prepared for the Directorate-General for Energy and Transport, Office for Official Publications of the European Communities, Tech. Rep., 2008, iSBN 978-92-79-07620-6.

[26] "Achieving energy efficiency through behaviour change: what does it take?" European Environment Agency (EEA), Tech. Rep., 2013, iSSN 1725-2237.

[27] "Key data on education in europe 2012," Education, Audiovisual and Culture Executive Agency (EACEA P9 Eurydice, EUROSTAT), Tech. Rep., 2012, iSBN 978-92-9201-242-7.

[28] D. Amaxilatis, O. Akrivopoulos, G. Mylonas, and I. Chatzigiannakis, "An iot-based solution for monitoring a fleet of educational buildings focusing on energy efficiency," *Sensors*, vol. 17, no. 10, p. 2296, 2017. [Online]. Available: http://www.mdpi.com/1424-8220/17/10/2296

[29] L. Pocero, D. Amaxilatis, G. Mylonas, and I. Chatzigiannakis, "Open source iot meter devices for smart and energy-efficient school buildings," *HardwareX*, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2468067216300293

[30] J. Pierce and E. Paulos, "Materializing energy," in *Proceedings of the 8th ACM Conference on Designing Interactive Systems*. ACM, 2010, pp. 113–122.

[31] S. Göbel, S. Hardy, V. Wendel, F. Mehm, and R. Steinmetz, "Serious games for health: personalized exergames," in *Proceedings of the international conference on Multimedia*. ACM, 2010, pp. 1663–1666.

[32] C. Petrioli, J. Potter, and R. Petroccia, "Sunrise sensing, monitoring and actuating on the underwater world through a federated research infrastructure extending the future internet," in *Proceedings of EMSO 2013*, Rome, Italy, November, 17 2013, demo also presented. [Online]. Available: http://www.emso-eu.org/management/images/documents/EMSO_ABSTRACT_final.pdf

[33] J. Baichtal, *Building Your Own Drones: A Beginners' Guide to Drones, UAVs, and ROVs*, 1st ed. Que Publishing Company, 2015.

[34] E. Mallon and P. Beddows, "The cave pearl project: developing a summersible data logger systemfor long term environmental monitoring," https://thecavepearlproject.org.

[35] "Green paper marine knowledge 2020, mapping to ocean forecasting," Luxembourg: Publications Office of the European Union, COM (2012) 473 final, Tech. Rep., 2012, iSBN: 9789279-25350-8.

Exploiting IoT Technologies for Personalized Learning

Evaggelos Spyrou
Institute of Informatics and Telecommunications
National Center for Scientific Research – "Demokritos"
Athens, Greece espyrou@iit.demokritos.gr

Nicholas Vretos
Information Technologies Institute, Center for Research and Technology Hellas
Thessaloniki, Greece
vretos@iti.gr

Andrew Pomazanskyi
Nurogames GmbH
Cologne, Germany
andrew.pomazanskyi@nurogames.com

Stylianos Asteriadis
University of Maastricht,
Maastricht, The Netherlands
stelios.asteriadis@maastrichtuniversity.nl

Helen C. Leligou
OTE Academy,
Athens, Greece
leligou@gmail.com

***Abstract*—This paper presents the IoT ready platform of the MaTHiSiS H2020 EU project. Sensing devices are used to capture the affect of learners during their interaction with learning material, which comes in the form of serious games although other forms are also considered. This interaction may use mobile devices such as smart mobile phones and tablets, but also robots. Within the context of MaTHiSiS, a learning process is broken down into "learning atoms", i.e., pieces of knowledge that may not be further divided. A set of learning atoms leads to a "learning goal", which is set by the tutor. The process of learning is non-linear, i.e., the order of learning activities that are presented to a user and are connected with a learning atom may be different per user. This personalization process may also have an influence in the difficulty of the learning actions and is modeled using the concept of the "learning graph". The overall system architecture complies to the IoT paradigm. A set of representative serious games developed for different use cases that exploit the available IoT infrastructure to personalize the learning experience is also presented.**

***Index Terms*—IoT, learning, affect recognition, serious games**

I. Introduction

During the last few years, advances in several research areas such as electronics, telecommunications and informatics have led to the so called "revolution" of the Internet-of-Things (IoT) [1]. The main idea of IoT is that daily used physical objects (i.e., the "things") are enhanced with the embodiment of short-range and energy efficient mobile tranceivers, which allow them to connect to the Internet. IoT derives from such an extensive networking and is expected to find numerous applications within a broad range of heterogeneous areas. Industry, logistics, building and home automation, smart cities and smart manufacturing are only few of the areas that have benefitted from the limitless opportunities [2] offered by this new research area which has been considered by some as the "next industrial revolution" [3].

IoT ecosystems typically adopt a service-oriented architecture (SoA) [4]. This means, that every part of the ecosystem whether being a hardware object (e.g., a measuring device) or some kind of software (e.g., an algorithm that processes the measurements of the device) is exposed to the outer world as a web service. More specifically, by web service we denote a self-contained unit of functionality, which offers its services to other "things" via the Internet. Such services may be typically categorized into three distinctive categories, based on their role within the ecosystem. More specifically, a) sensing services are used to capture some physical property (-ies) of the real world. In few cases they slightly process these measurements before they broadcast them; b) processing services whose role is to process and/or analyze the acquired measurements of the sensing services. Upon processing, they expose the inferred results; c) actuating services that enable certain actions, using as input the results of the processing services. Typically, applications that are designed with the goal to be deployed within an IoT ecosystem conform to the aforementioned SoA paradigm, involving distributed sensor networks at various scales, distributed processing modules and actuation elements.

On the other hand, serious games [5] within the classroom are considered to be the the next step in the educational/training environments. Recent developments mainly in the fields of informatics and hardware have made equipment such as a robot or even a tablet affordable to the masses. Although the ideas of serious games used for training exist for the last few decades, only recently they have been actually used within the classroom in mass education. Their main idea is to provide an attractive and motivating environment for their users so that the learning process becomes more effective.

Apart from the fact that students may prefer them over the traditional learning processes, they also have several significant advantages. Firstly, it is easier to acquire an assessment of the users' achievements. Secondly, they may adapt to the user's special needs, competences and performance more easily. Finally, in many situations they may be used efficiently even outside the classroom/training environment, since no special equipment is required. Since the aforementioned serious games typically require some kind of equipment to interact or observe (i.e., sense) their user, they may be developed to be compatible with IoT ecosystems, as previously described.

In this work we present the IoT-ready approach of the MaTHiSiS project, which is based on a platform that supports

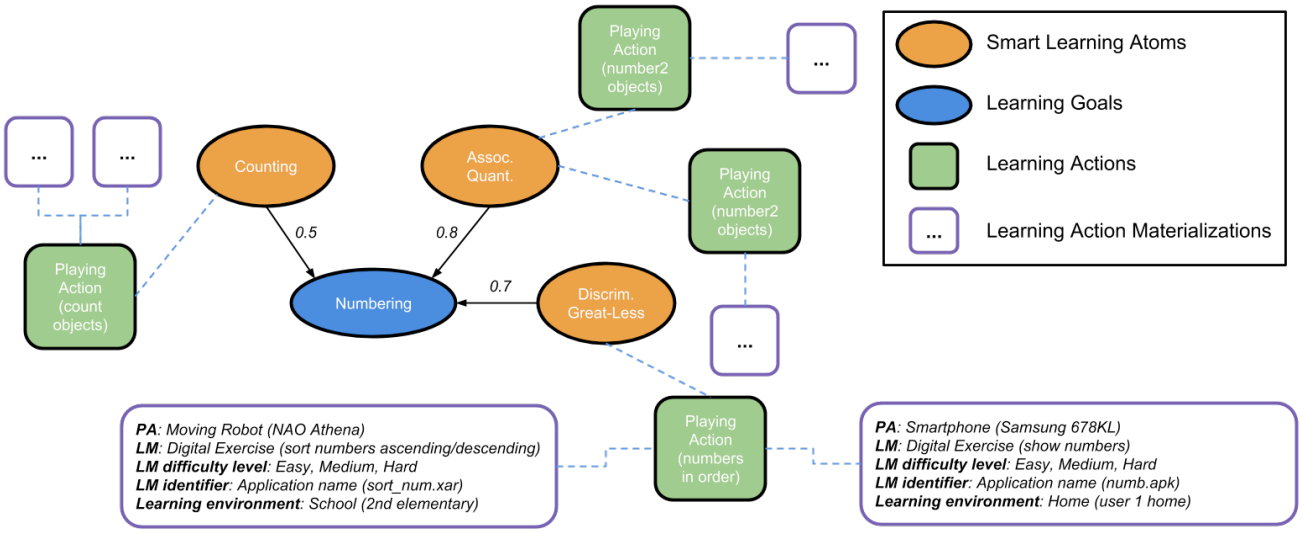


Figure 1: A learning graph

personalized, non-linear learning based on the users' affect state as collected by one or more sensors. The user is presented with some kind of learning material. A learning goal is set and the road to the achievement of this goal is divided into several tasks, which will be referred to as learning actions and are materialized as serious games. The user's affect state is continuously monitored by devices such as cameras, microphones etc., depending on the materialization.

The rest of this paper is organized as follows: Section II presents related work in the fields of personalization within serious games in education. Next, Section III describes the idea of personalization based on learning graphs, section IV presents the technical architecture of the MaTHiSiS systems that capitalizes on IoT to implement the personalization approach. In section V we present the serious games that adapt to the cognitive and affect state of the learner, while section VI provides valuable conclusions.

II. Related Work

The main innovation of our approach is personalization of serious-games based education based on affect detection in the wild. Existing solutions show a rapid development of the market. The most representative ones are: the personalized e-learning system using item response theory [15], adaptive i-Learning Management Systems based on standards (aLFanet) [16] and Tsal's progressive attentional training [17]. They offer a combination of machine learning techniques and user modelling for adaptive learning with multiple scenarios taking into consideration various personalization sources like previous knowledge, learners cognitive abilities and progress through the course. Likewise, a number of commercially available adaptive learning technologies on the market like Knewton, DreamBox Learning, Smart Sparrow are the examples of systems that capture learner's data and use learning analytics to facilitate tailoring of the learning paths. Although the solutions have already proven their effectiveness in terms of academic achievement of the learners as well as their users' satisfaction, they come with certain deficiencies: *first and foremost* they do not capture the temporal status of the learner, i.e. whether they are bored, engaged, frustrated, which has direct impact on the learning process evolution and relevant satisfaction leaving significant potential for improvement. Second, the majority of them requires the investment of significant amount of time for training the system and parameterization (which is not yet automated), still requiring a lot of manual and sometimes intensive design and codification tasks, which impedes rapid uptake of the adaptive learning systems in the teaching and learning communities. Another limitation is the fact that they are offered to a limited range of customer segments (due to limitations of technology to address specific needs of a wider group of segments, lack of domain knowledge).

In MaTHiSiS, we extend personalization to also take into account the learner's affect state using different IoT devices including cameras (available in laptops or robots), inertia sensors (available in tablets and smartphones) and micro-phones among others. Exploiting the camera recordings, facial expression analysis can be achieved. The facial expressions are often considered as the strongest indicator communication tool of human emotions. It displays people's feelings and mood state, from simple spontaneous emotions like happiness and disgust to time-dependent affective expressions states like anxiety, boredom and engagement in during a current task and/or a situation. The graph-based method's algorithm presented by D. Antonaras, et. al. in [6] exploits the ability of face facial image to be represented as a graph. A facial landmark detection algorithm locates specific areas of the faces using points of interest that are therefore used also to create the graph. Different emotions invoke different

movements of facial muscles and, thus, different positions of the points on the image yielding dissimilar graphs during an image sequence. Using this variation of the graphs, the algorithm predicts the different emotions per certain timeframes. The main steps of this algorithm are the face detection in the input image, the landmark (point of interest) detection, the feature extraction and finally, the facial expression classification. The performance of this algorithm using Cohn-Kanade [7] dataset to predict affective state reached a classification score that rounds up close to 100% accuracy.

With respect to inertia sensors, many recent studies support the potential usage of recognizing users' emotional states through various inertia sensors such as accelerometer and gyroscope [8][9][10][11]. Inspired by the research performed by Coutrix [12], an affect recognition system which exploits the expression through 2D and 3D gesture using aforementioned sensors can be implemented. In MaTHiSiS we expand this method focusing on the recognition of engagement, boredom and frustration. This three-dimensional and continuous space can be accurately mapped to affective states from the theory of flow. The features extracted are analysed in order to detect common patterns which can allow the system to infer the affect state of the user. Ideally, these features will help in the identification of erratic movements or unexpected behaviors such as the lack of motion or interactions with the devices. This information could denote frustration or boredom respectively. Emotion recognition based on gaze and speech cues have also shown to lead to high accuracy [13],[14] and are employed in MaTHiSiS system capitalizing on the relevant IoT services and the implementation of relevant algorithms in cloud infrastructures.

III. The MATHISIS Project

MaTHiSiS is an H2020 project that aims to promote a holistic approach to education, enabling seamless learning in a variety of learning environments and for different learner styles and needs. The learning vision is to provide a system that would be able to fit the needs of several educational/training contexts, ranging from mainstream education to vocational training. Moreover, it would be able to support both neurotypical learners, as well as learners with learning disorders. The pilot activities of MaTHiSiS cover both formal and informal education. Also, heterogeneous learning environments are supported. Users may interact with the MaTHiSiS platform inside the classroom, at home or even while commuting and may use a typical personal computer, a tablet, or even a smart mobile phone.

MaTHiSiS aims to provide a whole new ecosystem that will facilitate non-linear learning processes, which may be personalized in real-time and by considering the users' affective state. It consists of a fully integrated multi-agent,

interactive platform, which is complemented by a set of re-usable learning components. The platform is also able to automatically provide feedback and to assess a learner's progress and behavioral state. Games play a significant role within the whole process.

More specifically, a teacher (or tutor, in general) is responsible to define a non-linear learning experience. For such an experience, several learning objectives may be required. The platform is able to adapt the whole learning experience to the personalized needs of each user. The personalized process takes place into real-time, during the learning process. A learning goal is firstly set by the teacher. As learning goal, we denote the piece of knowledge, skill or competence, which should be acquired upon the learning process. In the context of MaTHiSiS, a learning goal may be divided into learning atoms, each comprising an atomic, yet complete piece of knowledge, skill or competence, which may not be further divided. Thus, a learning goal is the most primitive piece of knowledge towards a given learning goal. Note that a learning atom is *reusable*; a tutor may associate each with one or more learning goals.

The whole approach is graph-based (i.e., learning atoms comprise a graph) and allows the tutor to a) define the importance of the achievement of a given learning atom towards the ultimate learning goal, by setting the corresponding edge weights; b) to associate each learning atom with different learning actions and/or materials; and c) to adapt the weights of the nodes per user. Note that these weights reflect the personalized aspect of learning per user and are set based on her/his affect which is recognized by specialized sensors and processing units, as will be discussed in section IV. Overall, a learning graph may be seen as a learning scenario. Tutors are allowed to design their own learning graphs and are able to reuse learning atoms and learning actions, or define their own.

Adaptation (i.e., personalization) to a specific user is automatically performed. Her/his affect state is continuously captured while tutors are able to intervene at any time. Of course, personal needs such as disabilities or any kind of preferences are also stored in the users' profiles. Their emotional state is used to trigger the change of stimuli/learning activities provided. This way, for example, when a user is recognized to be in the state of "frustration," she/he may a) be guided to overcome the difficulty that may be the cause of this emotional state; b) be given the option to start a collaborative learning experience; or c) a different learning material may be presented to her/him. The overall goal is to keep the user engaged within the whole learning process.

An example of a learning graph is illustrated in Fig. 1. The learning goal is "Numbering." A tutor has defined that the goal may be achieved through a set of three learning atoms, namely: a) counting (Counting); b) association of numbers to quantities (Assoc. Quant.); and c) distinguishing between the notions "greater-than" and "less-than" (Discrim. Great-Less). As it may be observed, the largest weight is the one that corresponds to the discrimination, thus it is considered to have the most significant importance towards the learning goal. Note that in the presented case, each of the three learning atoms may be achieved by a single activity, which is actually a mini game. Given the competence level of the learner, a level of difficulty for the materialization of a learning action is decided by the platform. A personalized instance of the graph is therefore computed each time and weights may adapt to the competence achieved and the current affect state of the user.

IV. The Mathisis Architecture

The architecture of the MaTHiSiS platform has been carefully selected in order to realize the aforementioned vision which is based on the graph concept. It mainly consists of two sets of components that interact: a) components that are implemented within user devices which shall be referred to as platform agents (PAs); b) components that are implemented in the cloud, which shall be referred to as Cloud-based Learner Space (CLS). Within the pilot activities of MaTHiSiS, the PAs are laptop/desktop computers, smart mobile phones, interactive robots and robots. Through the PAs, users (depending on their role and the PA) may have access to authoring tools, platform configuration components, execution of learning experience and a simple UI for managing user accounts. The core elements of MaTHiSiS are depicted in Fig. 2.

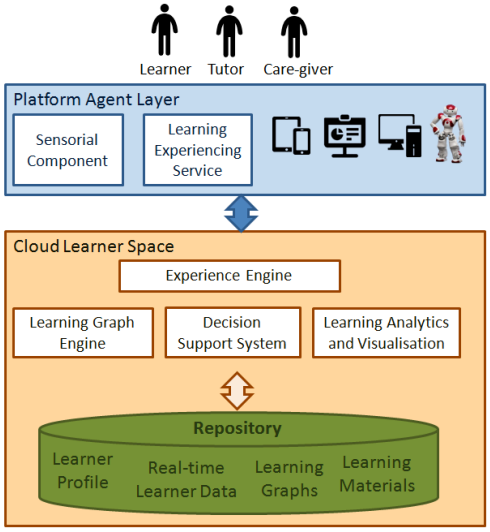


Figure 2: The architecture of MaTHiSiS

The core component of the MaTHiSiS ecosystem is the cloud learning space. More specifically, the learning experience, i.e., the execution of a learning graph and the materialization of the learning actions take place in the experience engine. The generated interactive content may take multiple forms, according to the location of the users. The decision support system provides and collects learning analytics as well as information regarding the users' affective state, so as to adapt and personalize to the users. The learning graph engine is responsible for the instantiation of the learning graph depending on the output of the decision support system. Finally, all information is stored within the repository.

The platform agent layer is instantiated into each device used, comprising by a UI between the user and the cloud-based learner space and also a component that is responsible for the execution of the learning materials; both consist the experience service. All sensors that are embedded within the user devices and are used for the detection of the affect state comprise the sensorial component. Their readings are sent to the cloud, and upon the detection of the affect, the graph may adapt to reflect the affect's new state.

The user roles within MaTHiSiS are tutors, learners and caregivers. Tutors are those that select the learning graph, define the learners and select the appropriate device for each, while the learning materialization is selected by the platform. Learners interact with the learning action materialization (while in the case of individual learning they also take the role of tutors). Finally, the role of caregivers is complementary to the previous two. They are responsible to prompt learners to interact with the learning material in cases where this is needed.

We should herein emphasize the analogy of the MaTHiSiS architecture to the one of a typical IoT ecosystem, as it has been described in section 1. Our system has sensors, which may be either integrated within the devices that host the learning experience (e.g., integrated cameras of laptop computers, accelerometers of tablets/smart phones) or are placed within the learning environment (e.g., cameras/microphones). Each sensor corresponds to a sensing service and is used as input to a processing service, i.e., the detection of the user's affect state, which is implemented in the decision support system. Actuation takes place by dynamically personalizing learning graphs and adapting to each user based on her/his affect and performance. Since all processing services run in the cloud, we consider the MaTHiSiS ecosystem as an example of an IoT-ready platform, which is planned to evolve to ensure scalability and provide seamless interconnection with any kind of sensing devices.

V. Serious Games within Mathisis

In this section we will present the implemented learning materials that have been developed in the context of the MaTHiSiS project and have been tailored to suit the specific needs of the devices that are available for the pilot actions. Currently, materializations that make use of a robot (i.e., the robotic layer) and also mobile devices (i.e., the mobile layer).

Figure 3: A happy avatar congratulates the user

A. The Robotic Layer

1) Games that are based on the Turtlebot robotic platform

A Turtlebot[1] robotic platform is used along with learning goals which aim to improve the students' mathematical skills, literacy or language comprehension. More specifically, the first learning materialization, i.e., the improvement of *mathematical skills* uses the robot to ask the learner to identify the largest/smallest number between a series of numbers that are presented in the screen and given on cards. An avatar depicted

[1] https://www.turtlebot.com/

on the robot's screen asks the learner which of a set of three numbers (which may be five/eight based on the difficulty set by the tutor) is the largest and which is the smallest one. The learner shows one of the cards, which corresponds to her/his answer. If the answer is correct, then a cheerful sound is played, the robot moves horizontally and the avatar's face changes to a happy one (Fig. 3). When the answer is wrong, a sound indicative to a failure is played. The robot moves rotationally and avatar's face changes to a sad one. The same series of events takes place in case the user does not respond, within a predefined amount of time (timeout). The interaction continues for a number of times, based on the difficulty and upon the completion the score is sent to the decision support system. Another similar application is also used to improve the learners' mathematical skills, by asking them to provide a series of number in order. The third application is used to associate numbers to quantities.

a. b.

Figure 4: Learner showing a card representing a syllable and a screen of the Turtlebot guiding the word composition activity.

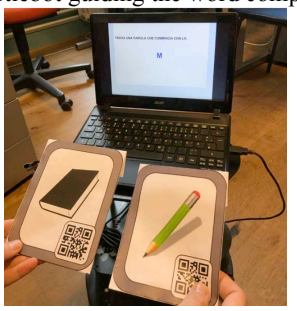
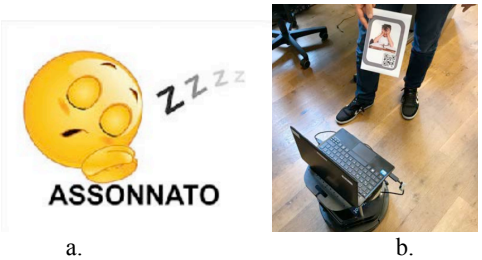


Figure 5: Turtlebot guiding the word recognition activity

Figure 6: Turtlebot following the learner during the word recognition activity

As for the improvement of *literacy*, the goal is to teach learners about word composition or association of words with the characteristics of objects. This application allows the learners to construct words with cards that depict syllables. The robot shows a picture of an object (Fig. 4b) and provides the learner a set of syllables, which should be used to compose the word that represent the object. The learner shows the card (Fig. 4a) in some order and the avatar then provides positive or negative feedback after each card is shown. When a correct syllable is shown, the word to be represented is built accordingly on the screen to guide the learner. When the word is finished, feedback is provided and the kid is congratulated by the robot. Another application is used for word recognition. The robot shows a letter and then the learner is asked to pick up the picture of the object starting with that letter among the provided cards (Fig. 5). The objects can be located in different parts of the class and the learner will look for them. During this activity, the robot follows the learner (Fig.6). QR coded are attached to objects, so that the robot will be able to confirm them. Another activity consists in finding what two pictures have in common, the solution is chosen among given options (available cards). Firstly, the robot shows two pictures. Meanwhile, the learner has several cards with words available and chooses the one that represents a term that both pictures have in common. Feedback is provided by the robot, which in case of success, congratulates the learner at the end.

Another set of activities aims at improving the *language comprehension* of the learners, in addition to reading skills. Firstly, the robot shows a sentence, asks the learner to read it and to show the corresponding picture among those provided. In order to perform the activity, the learner should visit the picture where the situation described in the sentence is represented. The pictures are placed in the walls of the class. An initial learning activity is included to evaluate reading and comprehension skills. During this phase, the learner is asked to read the sentence in the screen and then, the robot notifies if the learner read the sentence correctly, based on teachers' evaluation. Then, the learner has a set of pictures available in the walls of the class and one of them represents the situation described in the sentence. In case of success, the avatar congratulates the learner at the end. A "fill the gap" activity has also been implemented as part of the same learning goal. The robot first asks the learner to complete the sentence choosing the correct word among those provided, by showing an incomplete sentence. The learner has a set of cards with words to complete the sentence. Furthermore, an activity to associate words to a concept is available. The robot shows pictures and written words and asks the learner to match them and read the word while it used.



a. b.

Figure 7: Turtlebot's avatar representing a feeling and a card shown by the learner as part of the learning material

Emotion recognition is a challenging task for children and especially for people suffering from autism spectrum disorder. To this goal, two different activities have been designed, aiming to aid on the improvement of this skill, by training the learner to differentiate among different emotional states based on face expressivity. The first activity allows learners, especially those with autism, to train their emotional skills and improve their capabilities to recognize emotions. First, the robot displays pictures of people/cartoons expressing different emotions to the learner. Under each picture, the name of the emotion appears (Fig. 7a). Then, the robot randomly shows one of these pictures (without the corresponding label). The learner has cards with the possible emotions (the ones which were initially shown) and has to select the correct one (Fig. 7b). Feedback is provided by the robot, which congratulates the learner at the end, in case of success. A variant of the previous activity has also been implemented. In this activity, the learner has to recognize the emotion without previous help. First, the robot displays a picture of a person/cartoon expressing a concrete emotion. The learner has a set of cards with the possible emotions and has to select the correct one. Feedback is provided by the robot, which in case of success congratulates the learner at the end. Finally, a set of activities have been designed to improve motor/coordination skills. In that case, the applications defined require functionalities which are not available in this platform agent. However, taking advance of the re-usability of learning material that has been implemented in MaTHiSiS, this can be materialized through learning materials that have been developed to be used in other platform agents. These applications are web-based games that can be deployed in computers, tablets and smartphones. TurtleBot uses these applications and enriches them by providing an added value, including visual feedback through the movements of the robot.

2) Games that are based on the NAO robot

A NAO robot [2] has also been used in the context of MaTHiSiS. Most learning materials that are used with the NAO as platform agent depend on verbal or visual interaction of the robot with the learner.

Within the *Emotion Recognition* learning goal, the robot asks the learner to pick a number of cards depicting faces, to show them to it one by one and say the corresponding emotion. The cards that are placed in front of the learner depict faces with distinct expressions. Depending on the level of difficulty the faces may be sketches, drawings or photographs of people. The robot verbally explains the game to the learner, if she/he has not played it before. The learner taps the robot on the head and the game begins. The learner picks a card and shows it to the robot. The robot asks for the name of the emotion that the card shows. The robot gives feedback to the learner and after three iterations the game finishes and the robot responds accordingly. Similarly, an activity used for the identification of emotional facial expressions slightly differentiates than the previous one; the robot says an emotion and the emotion and the learner has to pick the card that depicts this emotion.

The first activity designed to improve the learner's *math skills* requires comparison of numbers. Note that the range of the numbers depends on the level of difficulty. Firstly, the robot explains the game to the learner, if she/he has not played it before. Then, the learner taps the robot on the head and the game begins. The robot says two random numbers, asks which is smaller or greater and the learner responds verbally. The robot gives feedback to the learner and after three iterations the game finishes and the robot responds accordingly. The second activity consists of sorting the numbers. The robot says five random numbers and asks the learner to repeat them one by one in ascending order. Again, the range of the numbers depends on the level of difficulty.

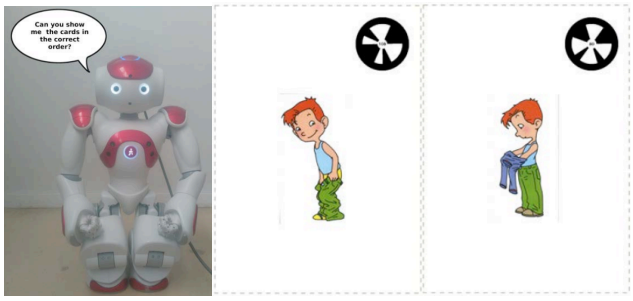
Another goal is to improve the learner's *motor skills*. The first activity requires the imitation of motor sequences. The robot performs a sequence of gestures and asks the learner to repeat them in the same order. Note that due to the lack of action recognition of the NAO robot, this material is assisted by the tutor. Again, the robot explains the game to the learner, if she/he has not played it before. The learner taps the robot on the head and the game begins. The robot performs an action. The complexity of the action depends on the level of difficulty. The learner performs the action, imitating the robot. The tutor taps the robot on different sensors depending on the learner's response. Finally, the robot verbally gives feedback. The second activity aims to train the spatial perception of the learner. First, the robot asks the learner to touch a part of its body and with the use of its sensors, it then recognizes the response and gives appropriate feedback to the learner.

A series of activities has been designed to improve the learner's *language skills*. Within the first activity the learner has to find a synonym of a word that is said by the robot. The options are also given verbally by the robot, but only one of them is correct. As the difficulty level rises, the words that are chosen are conceptually harder. The robot explains the game to the learner, if she/he has not played it previously and the learner taps the robot on the head and the game begins. The robot says the word and also says the three possible synonyms of the first word. The learner responds verbally and the robot gives feedback to the learner. After three iterations the game finishes and the robot responds accordingly. A slight variation uses antonyms instead of synonyms. Another activity aims to improve the learner's skills regarding semantics. In brief, the robot talks about a topic and then it asks the learner to say a word that is relevant to this topic. A tutor-assisted activity aims to improve spelling skills. The robot asks the learner to write down a word. As it is not possible for the robot to recognize different handwriting styles, the tutor responds if the learner spelled it correctly or wrong. An activity to improve the learner's vocabulary begins with the robot saying the names of various objects and asking the learner to match the names with the pictures that are laid in front of her/him. The difficulty level defines the number of cards used. More specifically, upon verbal explanation of the game by the robot, the learner taps the robot on the head and the game begins with the robot saying the name of an object. Then, the learner picks a card from the pool and shows it to the robot. The robot recognizes the marker on the card and gives feedback to the learner. After three iterations the game finishes and the robot responds accordingly.

[2] https://www.ald.softbankrobotics.com/en/robots/nao

A variation of this activity requires matching objects to pictures and repetition of their name. This learning material is actually the same as the one previously described, with the addition that the learner is asked to repeat the name of the object after showing the card to the robot. Within another variation, the robot says the name of an action, e.g., "wake up" and the learner has to pick and show the correct word card to match it. Alternatively, the robot says a word and then asks the learner to pick the card that has the word written on it. An activity to evaluate the pronunciation of the learner starts with the robot saying a word and then asking the learner to repeat it. As the difficulty level increases, the number of syllables in the requested word increases accordingly. Finally, in a variation the robot says an incomplete sentence and the learner has to say the missing word.

To improve the learner's *attention skills*, the first activity requires matching sounds to emotions. The robot plays a sound that expresses a basic emotion (e.g., laughter, crying etc.) and then it asks the learner to say the name of this emotion. Then, the learner has to say the name of the basic emotion that the sound represents and the robot responds accordingly, depending on the correct or wrong answer of the learner.



a. b. c.

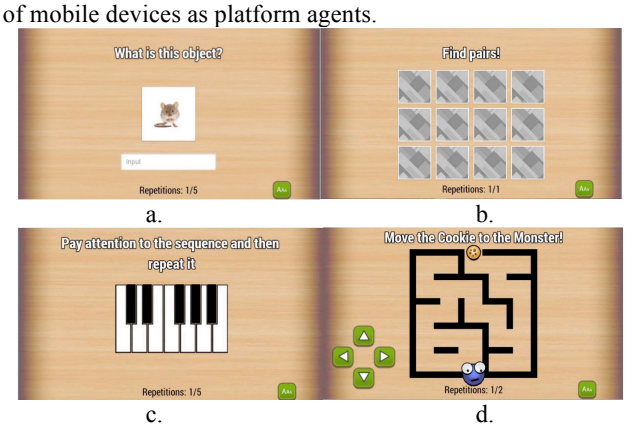
Figure 8: NAO asking the learner during the interaction and examples of two cards corresponding to the learner's answers

In order to improve *sequencing* skills, another activity requires sorting of pictures into logical order. Cards depicting various actions are placed in front of the learner and the robot asks her/him to show them to it in the correct logical order (Fig. 8a); e.g., the cards may show a child waking up, dressing, eating and leaving the house (Figs. 8b, 8c). The level of difficulty defines the number of cards in the sequence. When the learner shows the cards, the robot gives feedback after each card by recognizing the marker on each. A variation requires the learner to pick cards that are placed on the floor one by one in order to construct a correct sentence. Each card has one word, and the number of cards is dependent on the difficulty level.

Finally, *navigation* is another skill that the designed activities are planned to improve. The first activity aims to the identification of left and right. The robot raises one of its hands and then it asks the learner which hand was the one that it raised. The robot stands up. Depending on the difficulty level the robot may turn its back to the learner or keep facing her/him. The learner has to reply verbally and the robot gives feedback to the learner. Recognition of left and right direction is similarly materialized. In this case, the robot points to a direction and then asks the learner to say which direction it pointed to. In case of learning to turn left or right, the robot asks the learner to make it turn, using an external application that is provided, to the left or to the right side. This learning material requires the tutor to place one marker to each side, next to the robot. The learner performs this action using the external application and the robot recognizes the marker and provides feedback. A more advanced activity requires the learner to match pictures to rooms. More specifically, the robot says the names of various rooms that exist in a house and asks the learner to match the names with the pictures that are laid in front of her/him. The difficulty level defines the number of pictures that are placed in the pool. The learner picks cards from the pool and shows them to the robot, which then recognizes the marker on each card and provides feedback to the learner. The last activity requires the learner to navigate it, so that she/he finds a certain location, which is asked by the robot (e.g., table, backboard, etc.). Navigation is performed using an external application Markers need to have been placed in advance at the desired locations. When navigation is finished, the robot gives feedback to the learner

B. The Mobile Layer

Within MaTHiSiS, the role of mobile devices such as smart mobile phones and tablets is crucial, since, contrary to the aforementioned robots, they provide an economical solution for schools and as well as for individual learners. Moreover, they are equipped with gyroscope and accelerometer, which may be used to sense the learner's affect during interaction with the learning material. Finally, they may be easily connected with the cloud, where processing takes places. Therefore, in the context of MaTHiSiS, several learning activities have been designed to exploit the specific properties of mobile devices as platform agents.



a. b.

c. d.

Figure 9: Examples of learning materials that use mobile platform agents

An activity for *enriching vocabulary and object recognition*, prompts the user to name the object that is depicted on the screen and provides instructions to fill in the word describing the picture (Fig. 9a). Further variations offer the selection of the appropriate picture among those that are proposed that matches the words on the screen. Apart from correct/wrong answer, the time it took to complete the tasks is recorded. For the learning goal of *memory improvement*, a set of cards are

shown to the learner while they must turn them over one by one finding the pairs with same images (Fig. 9b). This activity is provided in three difficulty levels, with an increasing number of cards. Score and number of correct guesses are recorded. As for the learning goal of *motor skills and memory*, the learner is presented a sequence of interactions on piano keys and the goal of the game is that she/he would repeat the sequence playing a melody (Fig. 9c). The game starts with the challenge pressing one key at a time gradually increasing the number of keys to repeat based on the difficulty at which the game is played. The recorded result falls into one of the following categories: failed, passed or mastered. In use cases of autism and learning disabilities, a labyrinth game activity helps learners to develop *hand-eye coordination*. The game levels are based on the difficulty of the labyrinths the learner must navigate using navigation buttons on the screen, through adding more complex turns (Fig. 9d). The learners' aim is to move a "monster" along the obstacles/walls of the labyrinth and make the monster eat the cookie placed at the exit of it. Success and time taken are recording.

Finally, we should herein emphasize that we have developed a learning material that assumes the interaction between two learners, situated in different classrooms using tablets or one using tablet and another using a computer. Emphasis is on collaboration therefore a home screen that emphasizes this is a team effort and not a competition between two "players" is required. For a collaborative game, the two learners may well come into the game with various levels of competency. Their level at the end could be separately calculated, by taking into account their individual performance and how much help they had from or gave help to their partner. If both learners have similar affect profiles at the end of the game, next presentation of collaborative material could be the same for both. The alternative scenario is that one ends the game in a different affect state. We assume here that if a learner is bored, it is because the level is too easy for her/him, so the bored learner should be prompted to help the other one. This also makes the assumption that when a learner helps someone who is less competent, she/he becomes more engaged.

VI. Conclusions and Future Work

In this paper we presented the IoT ready platform of the MaTHiSiS project which relies on several heterogeneous devices, which host the learning materials. It is able to provide from low-cost solutions to sophisticated interaction using robots. Overall, MaTHiSiS may be characterized as a system that provides individualized and personalized pedagogical contents, adapting in real-time to the emotional response and intellectual level perceived by sensors, in a way that cannot be implemented by a teacher alone.

Initial driver pilots in diverse use cases of Education (i.e., mainstream, learners with PMLD and within the autistic spectrun) and in 3 countries (i.e., Italy, Spain and UK) have shown the effectiveness of the overall approach, they system's usability and performance. Currently, a second round of pilots (assisted pilots) reaches its end. These pilot activities make use of a prototype integrating the aforementioned technologies and the goal is to work in almost real-life conditions and assess the user experience and the suitability of offered services. Initial results mainly from tutors indicate the effectiveness and the potential of the MaTHiSiS approach.

Acknowledgment

The work presented in this document was funded through H2020- MaTHiSiS project. This project has received funding from the European Union's Horizon 2020 Programme (H2020-ICT-2015) under Grant Agreement No. 687772.

References

[1] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The Internet of Things: A survey. Computer Networks, 54(15):2787 – 2805, 2010

[2] Bin Guo, Daqing Zhang, Zhu Wang, Zhiwen Yu, and Xingshe Zhou. Opportunistic IoT: Exploring the harmonious interaction between human and the internet of things. Journal of Network and Computer Applications, 36(6):1531–1539, 2013.

[3] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of Things (IoT): A vision, architectural elements, and future directions. Future Generation Computer Systems , 29(7):1645–1660, 2013

[4] Ala I. Al-Fuqaha, Mohsen Guizani, Mehdi Mohammadi, Mohammed Aledhari, and Moussa Ayyash. Internet of Things: A survey on enabling technologies, protocols, and applications. IEEE Communications Surveys and Tutorials , 17(4):2347–2376, 2015*)*

[5] De Gloria, Alessandro, Francesco Bellotti, and Riccardo Berta. "Serious Games for education and training." International Journal of Serious Games 1, no. 1 (2014).

[6] D. Antonaras, et. al., D4.2 MaTHiSiS sensorial component, (2017), available at http://mathisis-project.eu/

[7] T. Kanade, J. F. Cohn, P. Lucey, J. Saragih, Z. Ambadar and I. Matthews (2010). The Extended Cohn-Kanade Dataset (CK+): A complete expression dataset for action unit and emotion-specified expression, San Francisco, USA.

[8] Coutrix, Céline, *et al.* (2012). Identifying emotions expressed by mobile users through 2D surface and 3D motion gestures. Proceedings of the 2012 ACM Conference on Ubiquitous Computing. ACM.

[9] Kim, H. J. and Choi, Y. S. (2012). Exploring emotional preference for smartphone applications. In 2012 IEEE Consumer Communications and Networking Conference (CCNC) (pp. 245-249). IEEE.

[10] Kim, Mira, *et al.* (2013). A touch based affective user interface for smartphone. IEEE International Conference on Consumer Electronics (ICCE). IEEE, 2013.

[11] Amelynck, Denis and *et al.* (2012).Toward e-motion-based music retrieval a study of affective gesture recognition. IEEE transactions on affective computing 3.2 .

[12] C. Coutrix *et al* (2012). Identifying emotions expressed by mobile users through 2D surface and 3D motion gestures, in in Proceedings of the 2012 ACM Conference on Ubiquitous Computing

[13] Xucong Zhang, Yusuke Sugano,Mario Fritz and Andreas Bulling (2015).Appearance-based gaze estimation in the wild

[14] M. Papakostas, E. Spyrou, T. Giannakopoulos, G. Siantikos, D. Sgouropoulos, Ph. Mylonas and F. Makedon (2017). Deep Visual Attributes vs. Hand-Crafted Audio Features on Multidomain Speech Emotion Recognition. Computation 5(2), 26, MDPI.

[15] Hwa Young Chen, Bong Hwa Hong, C. Lee, "A Learning System using user preference in ubiquitous computing environment", Futuretech 2011, CCIS 184, pp. 240-247, Springer

[16] Orga Santos, Carmen Barrera, Jesus Boticario, "An overview of aLFanet: an adaptive iLMS based on standards, AH2004, LNCS 3137, pp. 429-432, 2004, Springer-Verlang.

[17] L. Shalev, Y. Tsal, C. Mevorach, "computerized progressive attentional training (CPAT) program: effective direct intervention for children with ADHD", child neuropsychology, 13, (2007) pp. 382-388.L.

InLife: Combining Real Life with Serious Games using IoT

Pavlos Kosmides, Konstantinos Demestichas, Evgenia Adamopoulou, Nikos Koutsouris
Institute of Communication and Computer Systems
National Technical University of Athens
Zografou – Athens, Greece
{pkosmidis, cdemest, eadam}@cn.ntua.gr, nkoutsouris@telecom.ntua.gr

Yannis Oikonomidis
Synelixis Solutions SA,
Peissou & Chalkidos, Athens, Greece
oikonomidis@synelixis.com

Vanessa De Luca
SUPSI University of Applied Sciences and Arts of Southern Switzerland
Canobbio, Switzerland
vanessa.deluca@supsi.ch

***Abstract* - During the last 10 years, gamification has received increasing attention targeting a variety of people including children, students, youngsters and employers. In addition, great progress has been also observed in the Internet-of-Things (IoT) triggering various researchers' interest. In this paper, we present the core integration architecture and a serious game use case that are both implemented by the InLife project to drive new learning scenarios. InLife is European funded project that focuses on an innovative gamification framework targeting both typical as well as special education and social inclusion activities based on Serious Games. The core concept leverages on the potential of the IoT paradigm to link closely actions, decisions and events happening in real-life with in-game educational progress and modern gaming technologies. This bridge strengthens the infusion of gamification into non-leisure contexts, boosting at the same time the creation of new educational methodologies as well as new business opportunities.**

***Index Terms*—Internet-of-Things, gamification, sensors, serious games**

I. Introduction

Games are changing the way people learn, helping them think differently, and stimulating new ways in which people of all ages can use their minds. From empirical studies, Serious Games have proved to be effective in changing user behaviour models in real life [1]. Research is showing that games constitute a more interactive and participatory way to enable people of all ages better understand almost anything – from a history lesson to the dramatic change a flood can have on a specific community. They are increasingly used whether in formal education or at home, and also for vocational training. One can use a "virtual world" as a safe environment to try out certain behaviour and train repeatively that behaviour until the best approach to reach a certain objective has been learned. Simulation serious games have been widely used in many different fields, even in military or medical environments in order to train the behaviour of learners in specific situations [2].

The realisation of the potential benefits of serious gaming when used as educational tools in a sound pedagogical or social inclusion framework has recently started to be appreciated by the traditional computer gaming industry, now eager to branch out into new market segments [3]. Certainly, there has been a significant increase in activity within the research community but also within the enterprise sectors, with a wide number of companies starting to emerge in serious game market. Indicatively, the serious games industry was valued at slightly more than $2.5 billion in 2015, and is expected to more than double, reaching almost $5.5 billion by 2020 [4]. However, any company targeting the development of non-leisure games faces significant practical and pedagogical challenges. The most critical issue is that the modular tools currently available to the gaming industry are almost exclusively based around leisure-based gaming and do not support, or cannot be easily integrated, into educational contexts. Simultaneously, tailoring serious games to specific learning objectives poses a challenge for educators, as existing serious games generally do not allow easy adaptation of the content to the educators' own purposes.

In the frame of InLife, the focus is to facilitate the integration of real-world information into the game world and validate how this approach helps towards creating immersive, pervasive serious games that can have a significant impact in their intended educational character efficacy. This will be examined on the basis of the implementation of a gamification platform and the accompanied development of serious games, in which users can progress by completing specific tasks in real life, such as switching off a light when leaving a room or cooperate with other persons in simple actions, granted monitoring infrastructure by sensory devices implementing the Internet-of-Things (IoT) paradigm. InLife defines a hierarchical, multidisciplinary design approach (Fig. 1), bringing together research and applied expertise from several scientific fields, including gamification, pedagogical and sociological approaches, multimedia, computer graphics, human computer interaction for providing a comprehensive pervasive gamification development and integration environment.

InLife aims at producing, piloting, validating and demonstrating a novel, event-driven serious gamification framework for educational and social inclusion purposes, which directly links in-game progress and user experience to

real-life actions and decisions, detectable through an IoT infrastructure. The central novelty of the emergent concept is that serious gaming will be directly associated to the real world. The real-life actions will be detected by processing information coming from smart environments (smart metering and smart sensors installations) creating a bridge between the emerging IoT world on one hand, and Gamified virtual worlds on the other, enabling a multitude of educational, motivational and social inclusion applications.

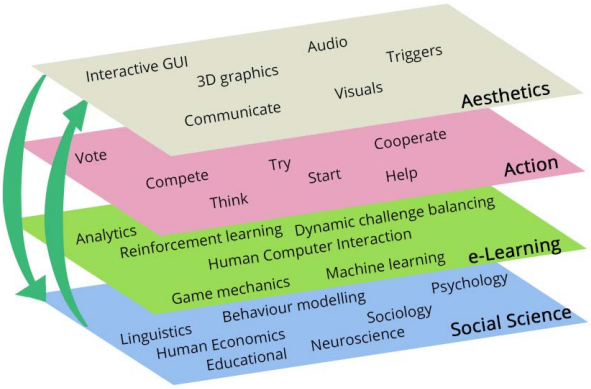


Fig. 1. InLife in a nutshell

It is essential to highlight that the InLife framework will be open and reusable, so that third-parties will also be able to create and/or configure their own 'in-life based' serious games, without having to start from scratch. Hence, through its work, the project will enable an ecosystem of InLife-based serious games and solutions multiplying the impact of the initial investment.

In the next sections, a high-level overview of the system components is presented herein, while a detailed presentation of the technical parts is out of the scope of the present paper. The rest of this paper is organized as follows. In Section II, related work is analyzed. In Section III, we describe the architecture of the proposed system, while in Section IV we present a use case Serious Game that was implemented. Finally, the paper is concluded in Section V.

II. State Of The Art

The InLife system intersects with several research areas and industrial sectors including (indicatively): Future Internet and the Internet of Things, sensor networks, especially using wireless protocols, social networking, gamification and serious games development. In the following sub-sections, we elaborate on these fields.

A. Social Media and Gamification in the context of INLIFE

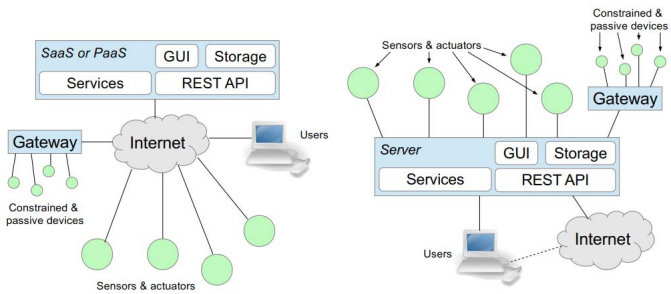
Social Media and online Social Networks in particular, have a major impact in everyday life. Facebook [5] and Twitter [6] are the most popular ones, but other types of social media are growing, focusing on a variety of themes, including reviews and ratings, blogging and conversations, location, DIY, wikis and business networking. Social media have been used in (serious or not) gaming platforms, mainly as a means to report gameplay progress in order to increase motivation and competition among players. This enhances the playing experience of users and contributes to the overall success of a game.

An example of using social media in a serious game involves taking into account the level of participation of the player in a social network (such as counting shares, likes, friends, commits, etc.) and allowing progress only if a checkpoint has been reached. In an educational serious game for instance, it could be asked that the player completes a task and then uploads a corresponding video on YouTube, where it gathers views by other players. The game engine could check that the user has indeed uploaded a video on their account and it has enough views to consider the task as fulfilled, providing the relevant reward.

In the context of InLife, social media have a much more significant role. InLife allows the tighter integration of social media into the gameplay, considering them not just another means to disseminate the results of playing the game, but also a source of information and a major parameter that affects the progress of the game. This notion is based on the fact that social networks are inherently collective and can therefore play a quite supportive role in games which focus on collaboration between users. The InLife framework will facilitate the effortless development of games that are tightly linked to social media, allowing these concepts to be used in practice.

B. IoT Management & Control

When building an IoT architecture, one of the major design choices to be made that affects usability and control is whether the platform should be local or cloud-enabled. These two approaches are depicted in Fig. 2 [7].



(a) Cloud-based platform (b) Local platform

Fig. 2. Typical IoT Platform architectures (Source: [7])

In the InLife concept, wireless sensor networks provide a basis for the IoT layer. Because sensors are a major input to the overall system, it is highly important that versatile, proven and effective technologies are used for the communication of sensor measurements. Two of the most popular wireless connectivity protocols deployed in this context are Zigbee [8] and Bluetooth Low Energy [9].

Sensors provide the required inputs to the system that can be used to detect events or confirm events registered into the

system manually by the users, in order to ensure that users are fair. This concept suggests a tight integration of usersourced information and sensor-based data. A simple example for this would be a kindergarten where the teacher asks the children not to make a lot of noise. Wireless sensors placed in the facility could monitor noise levels created by the children and objectively give feedback to them and the teacher in a game-like manner. This approach to enforcing the desired behaviour would be not only effective, but also entertaining for children, which would act collectively in order to achieve a certain group goal. IoT research and technology currently available present a solid basis for InLife to build upon, allowing us to create successful value-added services for IoT platforms.

C. *Using serious game concepts to influence behaviour change*

Gaming started out as a branch of children's play. Throughout history, children's games have brought up generations and have been widely recognized as a vital aspect of the development of one's personality, social skills, knowledge and other characteristics. However, playing games has not always been considered an educational activity and in the past was predominantly focused on young people as opposed to adults. This has changed in the recent past, where the concept of "serious games" was introduced. Serious games started out as a concept focused on [10] education through experimenting and investigation, having entertainment as a secondary purpose. Board and card games are two examples of serious games which have been around long before the introduction of electronic games.

One of the greatest challenges in the creation of serious games is to find the right proportions between efficacy and pleasure. A game of this kind should be effective in building skills, knowledge and competences for its players, while at the same time providing an acceptable but not excessive reward level. The effectiveness of such games has been proven [11] in the domain of industrial and military role-playing training games, and they have recently made their way into the educational domain. The success of serious games in education is based on some fundamental and inherent characteristics they possess; they are well-structured, highly motivational, they have a well-defined set of rules that are accepted by all participants etc. Of course, one of the most important aspects of serious games is the gameplay, which is what keeps the players interested in it. It should therefore be able to adapt to each participant's interest and time they spent playing the game, providing the necessary rewards, just as a traditional game would do. The importance of game design is further analysed below.

Behaviour modification, also known as applied behaviour analysis (ABA), refers to empirically derived techniques designed to influence the occurrence or frequency of certain behaviours. The research of behaviour modification dates back to 1911, when E. Thorndike frequently mentioned "modifying behaviour" in his article Provisional Laws of Acquired Behaviour. Since 1940s and 1950s, J. Wolpe [12] had adopted this term to describe psychotherapeutic techniques derived from empirical research. Common methods used in behaviour modification include increasing the adaptive behaviour through reinforcement, and decreasing the maladaptive one through techniques such as extinction, punishment or satiation, with emphasis on reinforcement measures.

In contrast to behaviour modification that imposes or removes stimuli to affect a behavioural change, gamification attempts to achieve the same result by creating an entertaining and engaging experience using the elements of a game or contest [13]. According to [14], gamification has four major elements that significantly increase its acceptance: increase of user satisfaction, conveyance of optimism, facilitation of social interaction and provision of meaning. In this way, compared to traditional behaviour modification methods (such as punishment), gamification relates behavioural or habit change to positive emotional feedback. InLife exploits gamification as a "technology" which aims to create entertaining experiences while accomplishing serious personal, social, or business goals. Gamification usually includes game elements such as leader board, rankings and points system, to create entertaining and engaging experiences. In addition, InLife uses advanced artificial intelligence algorithms based on ant-colony optimization in order to adjust the players' rewards based on their actions.

III. Generic Concept Architecture

InLife introduces a robust, integration and development framework providing the necessary ICT tools and services for building, simulating and validating interactive serious games and formal learning programs. InLife aims to implement an event-driven framework, where serious game evolution is tightly bound to real-life actions and conditions. To achieve this, it forces towards the following directions:

- Leverage on a reliable, modular and flexible IoT platform providing bi-directional interaction between gameplay activities and the surrounding real-space, taking advantage of IoT technology.
- Integrate data analytics, artificial intelligence and automation mechanisms, able to closely follow and analyse behavioural improvement and learning progress for each individual player and make decisions about triggering special learning actions when necessary.
- Realize a modular, flexible and open architecture that is able to i) integrate Serious Games in a wide range of educational learning and social inclusion contexts with zero or minimum external intervention and ii) operate under several different logics by adjusting critical serious game parameters, e.g. activate on demand audio/visual support, record and analyse players' information trails, formulate on the fly completion/cooperation clusters of players for certain purposes, etc.

The InLife architecture, depicted in Fig. 3, defines two major layers, namely the *IoT-based Data Adaptation Layer*, which establishes communication with smart devices and takes over data aggregation and adaptation, and the *Gamification Layer*, which coordinates InLife's services provision and gamification control. In InLife, trainees are able to access and play Serious Games through their portable smart devices, e.g.

smartphones and tablets, after they have been registered by the IoT platform.

In the following two subsections, the two major layers of InLife's open framework are thoroughly described both in technological and operational aspects.

A. IoT-based and Data-Adaptation Layer

Data aggregation in InLife is built on top of an open and flexible IoT Platform, which facilitates registration, communication, data flow and smart device management providing the core IoT infrastructure and services. The IoT platform implements both vertical and horizontal functions to support Gamification Layer applications. The essence of InLife's IoT platform is to enable the secure connection of a multitude of heterogeneous sensing and actuating devices, having different constraints and capabilities. This includes the interaction with the hardware infrastructure, including the control of smart meters, smart plugs and sensors. Indicatively, collected information will track peoples' or objects' mobility, lighting, temperature, room occupancy, pressure forced on objects/surfaces, location/acceleration measurements, interaction with smart objects, etc. The IoT platform provides the required scalability through its distributed message queue-based architecture for interfacing and collecting metering data from a large number of deployed meters. Also, it employs cloudification, service discovery and sophisticated data chain technologies, in order to define credible data adaptation and flexible data management mechanisms able to enable powerful administrative tools exploited by subsystems of the Gamification Layer. InLife's architecture also inherently supports different communications standards (mainly IEEE-based such as WiFi and ZigBee, etc.).

B. Gamification Layer

Gamification Layer is responsible to monitor and control Serious Game evolution and players' progress timeline integrating in a smooth and interoperable way the developed ICT-enabled automation and modelling components and services of the InLife open framework. It is built on a modular design, the main blocks of which and their corresponding responsibilities and functionalities are the following:

Context Information Modelling: The first step to demonstrate a successful Serious Game is to develop a comprehensive context design capturing all aspects of the (IoT-equipped) smart environment, within which the educational or social inclusion activity is unfold. Context information modelling (CIM) aims at identifying the firm and meaningful entities of the educational/social inclusion ecosystem and how each one supports the overall educational objectives. Further, it specifies what exact variations and degrees of freedom are available for each entity or part, as well as the relations between them both in a qualitative and quantitative aspect.

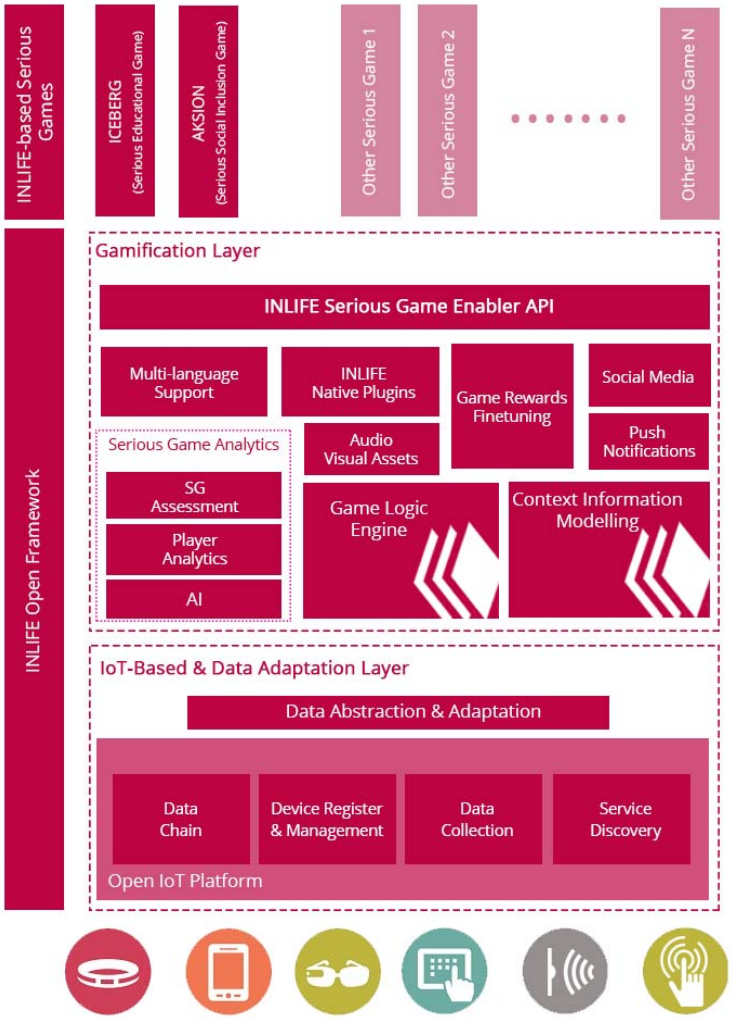


Fig. 3 InLife platform architecture

Game logic engine: In the context of Serious Games, this engine drives the development and integration of modules not traditionally associated with computer gaming but vital for non-leisure gaming contexts. Among others, this includes modules that provide support for bridging between the physical and digital world, interactive educational narratives, neurostimulation to promote learning and integration of haptic feedback into the learning experience, etc. The ultimate objective of the game logic engine is to define a number of alternative options for game structure and logic.

Serious Game Analytics integrates a number of artificial intelligence mechanisms and powerful analytics tools to understand in-game player behaviour and better measure overall Serious Game business success. The main goal is to provide advanced game analytics, while assuring functional consistency, data transparency and hiding any heterogeneity issue. The framework is able to seamlessly define and integrate sophisticated behavioural models for (group of) trainees by employing effective classification, feature extraction, clustering and time-series analysis over the collected data and discovering hidden relationships and inherent interdependencies. Based on these models and on regression and extrapolation techniques, the aim is to provide a better and more effective gaming environment for (group of) players by recommending both in-game adjustments and real-life actions, according to player's skills, response level in set challenges/goals and style of learning process. Apart from player-centric analytics, Serious Games Analytics will provide (actionable) insights on critical KPIs and welldefined metrics that outline the success of Serious Games, as a valuable means for all stakeholders to drive specific strategies/policies that improve their effectiveness, and to (re)train or remediate play-learners for performance improvement.

Game Rewards Fine-tuning fulfils two central needs: firstly, the serious game's need for optimizing its reward scheme to increase its effectiveness, and, secondly, the player's need for information that helps him/her to orientate, achieve goals and solve problems. Feedback about player's progress evolution and state is individualized (per player) leveraging on *Serious Game Analytics* and on three different mechanisms of feedback provision. Through intermittent and immediate feedback during the learning progress, the trainee will get reflections on what has been reached so far during game play. It is a key aspect of interaction that influences player's motivation and provides guidance and assistance when necessary. Based on serious games' Graphical User Interfaces (GUIs), text and texture objects are simple mechanisms to provide this type of feedback. Another kind of feedback would be the assessment and measurement of overall progress, which informs players about their overall level of performance and possible correlate it with other (group of) players, e.g. by showing scores tables, leader-boards etc. Finally, the third mechanism will utilize competitions and rewards to increase trainees' motivation and replayability.

Multi-language support which allows the quickly and easily change of language, as well as adding new languages to the user interface, to broaden the spectrum of potential users from different countries (also especially useful when addressing children of small ages).

Push notifications module that is responsible for broadcasting push messages to trainees, asynchronously announcing that new milestones have emerged or that certain milestones have been achieved. It may also be possible to display the achievements of peers / competitors in the serious game.

Social Media module that is responsible to activate interaction between Serious Game world and the social media. Hence, this module establishes a link able to upload information from Serious Game to the social media (e.g. leader-boards reports or rewards), whereas in the future (after the project completion) this link can be made bidirectional and also transfer actions caught from players' profiles in social media, public profiles or local news aggregators into the serious game in a context-aware manner.

InLife Native Plugins module that defines code libraries executed by Unity during the development process of Serious Games and allow developers to integrate particular middleware libraries of the platform or have access in context-specific features and attributes.

It is worth mentioning that InLife, through its open APIs, enables third-party developers to create and/or configure their own 'in-life based' serious games, without having to start from scratch.

IV. Use Case – Serious Game

In this Section we present the ICEBERG serious game which was implemented as a first use case scenario in InLife. ICEBERG is a combination of an online Role Playing Game (RPG) and a strategy game, which evolves on earth, but in the lost world of ice. The main creatures that live there are the Yetis, but other animals exist as well, like penguins, polar bears, albatrosses, whales. Each player has a Yeti to interact in the ice world and an "ICEBERG", which is the place where the Yeti lives. There are various types of resources in ICEBERG. The most important of them are the ice blocks, which are produced by penguins. The rate of this production per penguin will be configurable. It will also be possible to get access to various technologies and materials in order to erect buildings on the available space of the "ICEBERG". The screenshot below presents an indicative view of the ICEBERG graphical user interface.

Fig. 4 ICEBERG game interface.

When your behaviour is environmentally friendly and you are trying to be energy-efficient as suggested by the ICEBERG

game, then the area of your ICEBERG increases in the virtual world and at the same time you obtain more friends, namely penguins, polar bears, albatrosses, whales, that are coming to your ICEBERG and are willing to help you. For example, penguins are able to produce ice blocks, polar bears are able to create bridges and buildings, etc. A polar bear might become your friend when you complete a specific mission or achieve a target in the desired period of time. Similar settings will exist for the other animal friends as well. On the other hand, if the behaviour of the user is the opposite of what is expected, this can be depicted in the virtual world as well, e.g. the ICEBERG might start to melt, some penguins might get disappointed and leave, etc.

ICEBERGs can also be joined according to a set of rules. A hierarchical categorization of the ICEBERGs will be supported; the various levels can correspond to: individual player, room (office), floor, building, where e.g. a room-level ICEBERG represents the joining of all players in the same office room. Only ICEBERGs of the same level can be joined. The creation of such groups, i.e. local communities, is expected to be a very helpful feature for keeping the interest of the involved people to the game at high levels. Moreover, joining ICEBERGs at floor level could be done for distant buildings as well, since in the virtual world, distance is not a problem. Joining will be implemented as a game procedure that needs time and resources, e.g. ice blocks, in order to make a bridge. This can be used as a way to keep the players focused on a target at least for some period of time, which is something important, as new behaviour needs time (typically, a minimum period of three (3) weeks) and repetition in order to be established.

Central to ICEBERG is the notion of a gameplay directly linked to the real world. The main way to gain the necessary resources (animal friends, ice blocks, additional "ICEBERG" surface, etc.) for expanding within the game will be through specific actions in real life.

An extended list of example user actions that are of interest to InLife and can be rewarded, negatively or positively, is the following:

- Turn off the lights when leaving an empty room behind.
- Shut down the computer and other devices when they are not necessary.
- Use the stairs instead of the elevator, especially when going down.
- Close the windows when leaving the office, especially when heating or air-conditioning is required.
- Misuse the window shutters or blinds, by having them closed in a sunny winter day or open in a hot summer day, making the heating or cooling of the room more difficult.
- Select a room temperature as close to the external one as possible for heating or air-conditioning.
- Excessively use paper for printing.
- Throw paper, glass or plastic material to the regular bin instead of the recycle bin.
- Use of optical disks instead of a network to transfer data or instead of a cloud service to store data.
- Leaving the coffee machine always on, even if there is very little quantity of coffee inside.
- Borrow an e-book and not a paper book.

An initial beta version of the ICEBERG game has been released in Google Play (Fig. 4), accessible only from registered testers [15]. After the conduction of the foreseen trials, the final version will be distributed via Google Play, to all users.

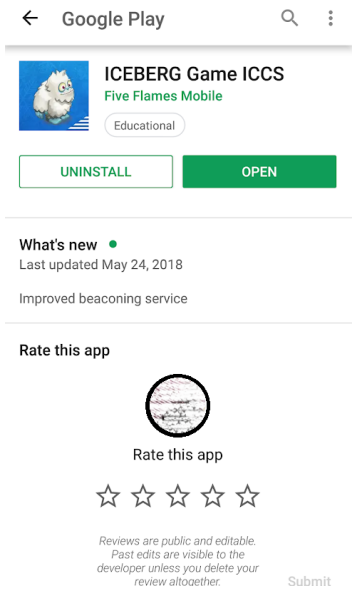


Fig. 4 ICEBERG game in Google Play

V. Conclusions

To summarize, the present paper presented the work conducted within the InLife project, which focuses on taking advantage of IoT technology and combine it with Serious Games. The core architecture was presented, as well as a serious game use case. The realization of the InLife vision will ultimately pave the way for the proliferation of new innovative IoT-based serious games, created also by third parties, featuring enhanced gameplays and educational efficacy, thus establising new market opportunities for involved stakeholders.

Acknowledgment

This work has been perfomed under the H2020 732184 project InLife, which has received funding from the European Community's Horizon 2020 Programme. This paper reflects only the authors' view, and the Community is not liable to any use that may be made of the information contained therein.

References

[1] R. Z. Enciso, "Simulation games a learning tool", Proc. Int. Simulation and Gaming Assoc. Conf., 2001-Sep , pp. 1-13.

[2] D. Schmorrow, and D. Nicholson D, “Advances in Cross-Cultural Decision Making,” CRC Press, 2010, pp. 77–86.

[3] J. Stewart, et. al. “The potential of digital games for empowerment and social inclusion of groups at risk of social and economic exclusion: evidence and opportunity for policy,” European Commission, Joint Research Centre, Institute for prospective Technological Studies, 2013.

[4] Rohan, “Serious Game Market worth $5,448.82 Million by 2020,” MarketsAndMarkets, Available online: http://www.marketsandmarkets.com/PressReleases/serious-game.asp (accessed April 2016).

[5] Facebook: https://www.facebook.com

[6] Twitter: https://twitter.com

[7] J. Mineraud, O. Mazhelis, X. Su, and S. Tarkoma, “A gap analysis of Internet-of-Things platforms,” Computer Communications, vol. 89, pp. 5-16, 2016.

[8] ZigBee Alliance. Available online: http://www.zigbee.org/ (accessed April 2016).

[9] Bluetooth Low Energy. Available online: https://www.bluetooth.com/what-is-bluetoothtechnology/ bluetooth-technology-basics/low-energy (accessed April 2016).

[10] C. Abt, Serious Games. New York: The Viking Press, 1970.

[11] P. Pivec, “Game-based learning or game-based teaching?,” Technical report, BECTA UK, 2009.

[12] J. Wolpe, “Psychotherapy by reciprocal inhibition,” Integratice Physiological and Behavioral Science, vol. 3(4), 1968, pp. 234-240.

[13] M. Petkov, and G.E. Rogers, “Using Gaming to Motivate Today's Technology-Dependent Students,” Journal of STEM Teacher Education, vol. 48(1), 2011, pp. 7-12.

[14] I. Blohm, and J.M. Leimeister, “Design of IT-based enhancing services for motivational support and behavioral change,” Business & Information Systems Engineering, 2013, pp. 275-278.

[15] ICEBERG game in Google Play for testers: https://play.google.com/store/apps/details?id=com.fiveflamesmobile.iceberg.iccs

Multi-Parameterised Matchmaking: A Framework

Anders Harboell Christiansen
Department of Applied Mathematics and Computer Science
Technical University of Denmark (DTU)
Lyngby, Denmark
andershc1993@gmail.com

Emil Gensby
Department of Applied Mathematics and Computer Science
Technical University of Denmark (DTU)
Lyngby, Denmark
emil@gbcnet.dk

Bo Friis Nielsen
Department of Applied Mathematics and Computer Science
Technical University of Denmark (DTU)
Lyngby, Denmark
bfni@dtu.dk

***Abstract*—The competitive scene in online video games is becoming more and more prominent and player satisfaction is of key importance when it comes to a good user experience and a successful game. As such it is important to have efficient skill rating and matchmaking systems in order to provide a proper match experience. We propose a mathematical framework for the analysis of matchmaking systems. The mathematical model addresses the estimated skill or rating, calculation of winning probabilities based on the estimated skill, and the updating of the estimated skill upon completion of a game. We will briefly apply the framework to the ELO skill rating system. Next we will use the framework to analyse the robustness of the TrueSkill algorithm and discuss some of the findings. We have used simulated data to test the robustness of the TrueSkill algorithm. All of the data processing has been done in Python using our own code, built-in functions and Python packages. The code has primarily been used to make the simulations of matches and customise updating functions.**

***Index Terms*—matchmaking, mathematical model, player satisfaction, online video games, skill rating, framework**

I. Introduction

The online gaming community has grown steadily since the commercialisation of the computer and today, the most popular online game, League of Legends, has 100 million people playing every month and at peak times they have 7.5 million players playing at once [1].

The large amount of players who want to play a game for fun or even to become good enough to compete in e-sports – the professional side of online gaming – has given rise to the idea of matchmaking; since different players have different skills, a game between two players of widely different skills is not considered interesting. It is thus the primary goal of matchmaking to ensure an entertaining match.

In competitor-versus-competitor games, this translates to finding two players of equal or close-to-equal skill, making the match both mathematically fair and also entertaining. However, in teams, the interplay between these two aspects of game quality can sometimes be opaque. An entertaining match is dependent on some kind of mathematical fairness, but how that fairness is defined can lead to very different matches.

We did a limited search for a formalised framework that was capable of analysing matchmaking algorithms, but only found a partial one in the TrueSkill paper [2]. In this article we will thus separate a matchmaker into three parts: A skill rating part, a winning probability calculation part and a matchmaking part. The skill rating part must satisfy the conditions of being able to represent a player's skill, updating the skill, and placing this skill on a leaderboard that determines a ranking of players. The winning probability calculation part should be able to use the estimated skill ratings to calculate the winning probability of any kind of match-up, such that the matchmaking part can match players to provide entertaining matches. To exemplify our work we will apply it to two well-known skill rating systems, the ELO rating system and the TrueSkill rating system.

II. Framework

We will now define a framework for a skill rating system based around a single match. The skill rating system will represent a player's skill, update a player's skill and place the player on a leaderboard. Let us first define some terms. Let n be the number of unique players and let the set $U = \{1, ..., n\}$ be the set of these players. Let ρ be the number of players involved in a single match and the set of these players be $Z \subset U$. Let k be the number of unique teams involved in the match and let the set $M = \{1, ..., k\}$ refer to each of these teams. Any team $Q_j, j \in M$ is of arbitrary size where the teams are subsets of the whole, $Q_j \subset U$, such that $Q_i \cap Q_j = \emptyset$ when $i \neq j$ and $\cup_{i \in M} Q_i \subset U$. Finally, we define the function $|\cdot|$ as the cardinality of the set – how many elements the set contains.

With these definitions in hand, we will now present the framework, starting with the representation of a player's skill. Here, we let the random vector S denote a random variable of some distribution that *is* the player's skill. Our best estimate

978-1-5386-4359-4/18/$31.00 ©2018 IEEE

of that player's skill before the match is then denoted E^b. The best estimate of that player's skill after the match is similarly denoted E^a.

With the available players in U and their estimated game skills, E_i^b, we must now create k teams of players from the subset, Z. These can be denoted as teams $\mathbf{E}_1^b = \left\{E_{11}^b, E_{12}^b, ..., E_{1x}^b\right\}$ through $\mathbf{E}_k^b = \left\{E_{k1}^b, E_{k2}^b, ..., E_{ky}^b\right\}$ as long as $\left|\cup_{j\in M}\mathbf{E}_j^b\right| = \rho$. Note here that team 1 has x players and team k has y. This is merely to indicate that some games incorporate a lopsided number of players such as the game 'Evolve', which has 4 players on one team and 1 player on the other team [3]. In the ordinary case, $x = y$, that is, there is an equal amount of players on each team. The process of creating the teams also transforms the estimated individual skill ratings into estimated team game skills, $D_j = l(\mathbf{E}_j^b)$ where l is called the matching function and $j \in M$.

We can now estimate the quality of the match through a winning probability calculation. The closer the teams are to having an estimated equal chance of winning the game, the higher the quality of the match. This calculation can be expressed as $W = c(\mathbf{D})$, where W is the estimated game outcome also called the ranking of the teams, c is the function that calculates this estimated game outcome given the estimated team game skills and $\mathbf{D}$ is the matrix containing the random vectors of estimated team game skills.

When we create the teams that gives rise to this estimated game outcome, several other elements of the framework come into play. For one, we must take into account that despite a player having a certain underlying skill S_i, that player does not necessarily perform optimally in the current match. We therefore introduce the individual game skill, G_i, a random vector with a conditional distribution based on the underlying individual skill parameters. The conditional density can be expressed mathematically as $f_G(g_i|S = s_i)$. Next, we gather the chosen players into their teams $\mathbf{G}_1 = \{G_{11}, ..., G_{1x}\}$ through $\mathbf{G}_k = \{G_{k1}, ..., G_{ky}\}$, thereby calculating the actual team game skill as $T_j = h(\mathbf{G}_j)$. Note that l is the matching function that uses the estimated skills to calculate the estimated team game skills, but h is the matching function that calculates the actual team game skills given the player's actual in-game performance (the individual game skills). Finally, the actual comparison of the T_j's then gives the actual game outcome, the ranking R.

We can now come back to calculating the updated estimated individual skill ratings with the new information gathered from the game outcome. For each player $i \in Z$, we can then express the update as $E_i^a = \Psi\left(E_1^b, ..., E_\rho^b, R\right)$ with Ψ being the update function. Ψ could potentially be used to alter the influence of the ranking in the update of individual player skills in team games. With the notation for the framework defined, we can now redefine the core property of a skill rating algorithm as an update of the estimated individual skill parameters based on the ranking of games, that is the conditional distribution $f(e^a|E^b = e^b, R)$. This posterior will then function as the prior for the next match. We will now apply these concepts to some known skill rating systems.

III. The ELO Rating

The ELO rating system is a well-known skill rating system. It was invented by Arpad Elo and implemented as early as 1960 in games of chess. An ELO rating is a number usually between 0 and 3000 [4] – although theoretically it could be higher – that describes a player's estimated skill. Note that we do not call the ELO rating system a matchmaker, but a skill rating system, since it does not directly involve matchmaking.

The ELO rating system builds on the assumption that a player's skill is a normally distributed random variable [4]. In our framework, that means S is a normally distributed random vector of length 1. For notation, we will use $X \sim N(\mu, \sigma^2)$, where X is the normally distributed variable and μ and σ^2 are parameters, the mean value and the variance respectively. We can then say that $S \sim N(\mu, \sigma^2)$. The estimation of this skill, E^b, is then the ELO skill rating.

The individual skill parameters then gives rise to that individual player's game skill, G – that player's performance in one particular game – as another normally distributed random vector of length 1. We can express that G has the distribution $G \sim N(S = s, \beta^2)$. Here, the mean value is the particular realisation of that player's individual skill s. β^2 is a variance parameter that is unique to which game is being played and determines the length of the so-called 'skill chain', that is, how much of a change in skill is necessary for a better player to have a much higher chance of winning a particular game. Going forward we will abbreviate the name – such as [2] has done – as the 'skill chain variance'.

In our framework, the next step would be to apply the mapping that moves the estimated individual skills of all players $E_i^b, i \in U$ onto teams 1, $\mathbf{E}_1^b = \left\{E_{11}^b, ..., E_{1x}^b\right\}$, through k, $\mathbf{E}_k^b = \left\{E_{k1}^b, ..., E_{1y}^b\right\}$, to get the estimated team game skills $D_j = l\left(\mathbf{E}_j^b\right), j \in M$. However, since ELO is designed only for competitor-versus-competitor matches [5], there are only two teams with one player on each team, which means $\rho = 2$ and $x = y = 1$. Further, the mapping l is just the identity function, which means $D_1 = E_{11}$ and $D_2 = E_{21}$ – the estimated team game skills is the ELO rating as well. This also means that the estimated ranking would be one of three possible outcomes; player 1 defeats player 2 (denote this event $W = 1$), player 2 defeats player 1 (denote this event $W = -1$) or the two players draw (denote this event $W = 0$) [2]. This range of W replaces the estimated ranking vector in our framework.

There is one more simplification to consider. The event $W = 0$ is pushed to the side as a special case when it comes to determining probabilities. This relates to the way the ELO rating system updates the individual skill parameters, where a draw is treated as half a win and half a loss [6]. This treatment means we can simply calculate the conditional probability that given player 1 and 2's skill parameters e_1^b and e_2^b, player 1's individual game performance exceeds player 2's individual game performance and vice versa [2]. Denoting Φ as the standard cumulative distribution function of the normal distribution, this corresponds to the probability of the event $W = 1$ and $W = -1$, which can be calculated as

$$P\left(W = 1 | E_1^b = e_1^b, E_1^b = e_2^b\right) = P\left(G_1 > G_2 | E_1^b = e_1^b, E_2^b = e_2^b\right) = \Phi\left(\frac{e_1^b - e_2^b}{\sqrt{2}\beta}\right) \quad (1)$$

$$P\left(W = -1 | E_1^b = e_1^b, E_2^b = e_2^b\right) = P\left(G_2 > G_1 | E_1^b = e_1^b, E_2^b = e_2^b\right) = \Phi\left(\frac{e_2^b - e_1^b}{\sqrt{2}\beta}\right) \quad (2)$$

In this distribution, the event $W = 0$ has probability 0. We denote the number between 0 and 1 that comes from equation 1 as the expected score of player 1. The value from 2 is then the expected score of player 2 and the reciprocal value such that the sum of the two scores equals 1. An expected score is defined as the long-term ratio of games that that player will win. An expected score of 0.5 is thus an expression of a mathematically fair game, where each player will win every other game on average. If we now denote the difference between the observed score and the expected score as Δ we can calculate the exchange of ELO rating between a winning player and a losing player as

$$\Delta = \alpha\beta\sqrt{\pi}\left(\frac{r+1}{2} - \tilde{\Phi}\left(\frac{e_1^b - e_2^b}{\sqrt{2}\beta}\right)\right) \quad (3)$$

where $r = 1$ if player 1 wins, $r = -1$ if player 2 wins and $r = 0$ if the game ends in a draw, α is a number between 0 and 1 that determines the weighting of newer matches compared to older estimates – with a small α leading to few fluctuations in skill and large α the opposite. The two player's skill parameters can then be updated as

$$e_1^a = e_1^b + \Delta \quad (4)$$

$$e_2^a = e_2^b - \Delta \quad (5)$$

Although it is outside of our framework, it should be mentioned that there are many different ways to tackle the initial prior of the population in ELO. Some systems simply assign an initial ELO rating to a player, whereas other systems give a provisional rating for the first 10 to 20 games before using these games to estimate an initial ELO rating [4]. Finally, we note that the ELO skill rating system mostly relates to the representation of a player's skill with the possibility of having a leaderboard. The only part that can be related to the winning probability is the expected score as calculated through equation 1. This equation could be used for estimating the winning probability of players with the intention of matching them. However, as mentioned, ELO has no dedicated matchmaking part. We will now turn to the TrueSkill algorithm.

IV. Simulation

We will do a practical application of the framework for the TrueSkill algorithm. We use a player pool of 1200 players and assign each player a TrueSkill rating with values $\mu = 25$, $\sigma = \frac{\mu}{3} = 8.33$, $\beta = \frac{\sigma}{2} = 6.25$ and $\tau = \frac{\sigma}{100} = 0.125$ along with a match counter and number of wins (at first, both of these are set to zero). The value of $\mu = 25$ and $\sigma = 8.33$ is the default and recommended value by Microsoft [7], but it can potentially be set to any value – it all comes down to the tweaking of the skill and how it fits the game. We also set the draw probability to 0.0% as in our simulations we do not allow draws to happen.

Next, we sample the players' real skill from a normal distribution with $\mu = 25$ and $\sigma = \frac{\mu}{3}$. These are then used when we calculate the winning probability, but we match based on the estimated skill. This fulfils TrueSkill's assumption of the player base being normally distributed and we can now test whether TrueSkill can recover the real skill with the TrueSkill matching function l_1. This matching function works on the principle that the average of one team's estimate game skill should be as close as possible to the other team's average. We play out matches with two teams of six players each. To simulate that these matches play out in real time, we disallow any player to have played more than 2 matches more than the player that has played the lowest amount of matches. As an example, say that a player has played 0 matches, then the player pool cannot contain any player that has played 2 matches.

After creating the two teams, we use an estimated winning probability calculation to ensure that the match quality is high enough. This is taken from Moser [8], despite TrueSkill not ordinarily having a winning probability calculation. If the estimated winning probability for one team is above 55 percent, we throw away the teams and create two entirely new teams. Writing up all the steps of this simulation as a list of bullet points we get:

1) We put together an optimal team by using the l_1 matching function
2) We check the quality of the match by using the estimated winning probability
3) If the quality is good enough, we determine which team wins based on their true underlying skill
4) The estimated skill rating of the twelve players are updated
5) We stop the simulation when all players have played more than 200 matches

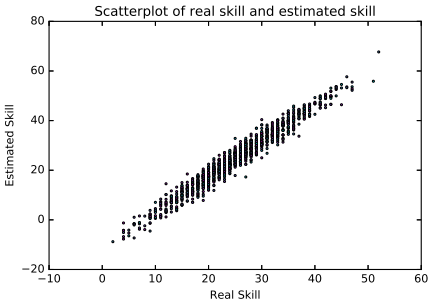


Fig. 1. Scatterplot with the normal data.

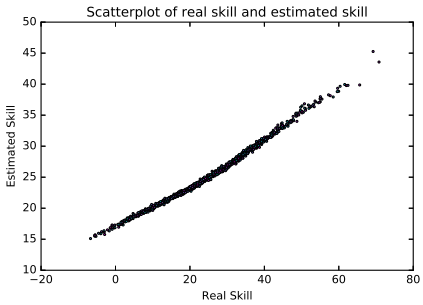


Fig. 2. Scatterplot with the log-normal data.

To compare the estimated skill and the 'real' skill, we can use a scatterplot, see Figure 1. Next, we calculate the correlation between the estimated skill and the real skill of the figure. The correlation of the scatterplot in Figure 1 is 0.98. This correlation and the shape of the scatterplot confirms that the estimated skill rating and the real skill ratings are close.

To assess the robustness of the TrueSkill algorithm, we next tried to change the underlying player base as having a log-normal distribution. This can be likened to a player base, such as the game of Overwatch, which has a slightly skewed distribution with a long tail that resembles a log-normal distribution [9]. As such, we next tested the TrueSkill algorithms' robustness to a log-normal player base. Using the same methodology of playing matches and checking the quality, we can compare the 'real' skill distribution and the estimated skill distribution of the player base after 200 matches via the scatterplot, see Figure 2. The correlation of this scatterplot is 0.99 after 200 matches. We have also tracked the evolution of the correlation for both the log-normal and the normal distributions as the matches are played as seen in Figure 3. Note that the x-axis is the total amount of matches played. It appears that the log-normal data actually outperforms the normally distributed data for this seed in the simulation.

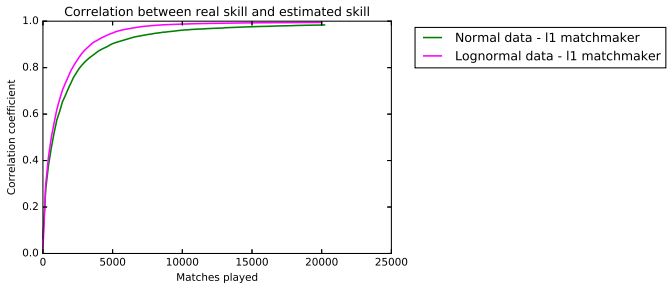


Fig. 3. Evolution of correlation between real and estimated skill distributions.

From this example, we can see that TrueSkill is relatively robust to the log-normal distribution. The above example can be considered an example of how to analyse the distributions of S and G and how they relate to the rest of the skill rating, winning probability and matchmaking systems.

V. Conclusion

In conclusion, we have begun work on a formalised framework for analysing skill rating systems. We managed to fit it with the ELO rating system, but we can also see that the framework requires additional work to fit more complex systems such as TrueSkill by comparing it to the partial framework presented in [2]. For the future, the framework could be expanded upon such that these more complex systems are also able to fit into the framework.

References

[1] P. Tassi, "Riot Games Reveals 'League of Legends' Has 100 Million Monthly Players", published September 13th 2016 on https://www.forbes.com/sites/insertcoin/2016/09/13/riot-games-reveals-league-of-legends-has-100-million-monthly-players/ , visited on: 15-03-2018

[2] R. Herbrich, T. Minka and T. Graepel, "TrueSkill: A Bayesian Skill Rating System", published January 1st 2007 on https://www.microsoft.com/en-us/research/wp-content/uploads/2007/01/NIPS2006_0688.pdf

[3] M. Gerardi, "Multiplayer matchmaking might be doomed to suck", published 20th February 2015 on http://www.avclub.com/article/multiplayer-matchmaking-might-be-doomed-suck-215567 , visited on: 15-03-2018

[4] M. E. Glickman and A. C. Jones, "Rating the Chess Rating System", published April 8th 2016 on http://www.glicko.net/research/chance.pdf , visited on: 15-03-2018

[5] D. Ross, "Arpad Elo and the Elo Rating System", published Fall 2007, Indiana University of Pennsylvania

[6] D. R. Brillinger, L. T. Fernholz and S. Morgenthaler, "The Practice of Data Analysis: Essays in Honor of John W. Tukey", published 1997 by Princeton University Press, p. 161

[7] T. Minka, "TrueSkill Ranking System", published November 18th 2005 on https://www.microsoft.com/en-us/research/project/trueskill-ranking-system/, visited on: 21-04-2017

[8] GitHub User 'jsnell' and J. Moser, "Win Probability?", published October 21st 2015 on https://github.com/sublee/trueskill/issues/1#issuecomment-149762508 , visited on: 21-04-2017

[9] S. Mercer, "Competitive Season 4 Blog - Follow Up", published March 3rd 2017 on https://us.battle.net/forums/en/overwatch/topic/20753625906 , visited on: 15-03-2018

Using Discrete Time Markov Chains for Control of Idle Character Animation

Adam Streck
Aging and Cognition
German Center for Neurodegenerative Diseases
Magdeburg, Germany
adam.streck@dzne.de

Thomas Wolbers
Aging and Cognition
German Center for Neurodegenerative Diseases
Magdeburg, Germany
thomas.wolbers@dzne.de

***Abstract*—The behavior of autonomous characters in virtual environments is usually described via a complex deterministic state machine or a behavior tree driven by the current state of the system. This is very useful when a high level of control over a character is required, but it arguably does have a negative effect on the illusion of realism in the decision making process of the character. This is particularly prominent in cases where the character only exhibits idle behavior, e.g. a student sitting in a classroom. In this article we propose the use of discrete time Markov chains as the model for defining realistic non-interactive behavior and describe how to compute decision probabilities to normalize by the length of individual actions. Lastly, we argue that those allow for more precise calibration and adjustment for the idle behavior model then the models being currently employed in practice.**

***Index Terms*—DTMC, AI, animation, virtual agents**

I. Introduction

Games and simulations often require autonomous decision making on the level of non-player characters (NPC) that would create an illusion of independent thought. The result of such a decision then manifest itself via an animation that is constructed based on the underlying simulation strategy, as illustrated in e.g. [1]. This is usually governed by a decision making system, which based on a certain strategy selects an action, executes the given action, waits until it finishes, selects the next action and so on [2]. This decision process is typically modeled either as a state machine, a behavior tree, or a utility function [3]. This is arguably a good solution for a sequence of actions with effects, e.g. walk to a door, open the door, walk through the door, close the door. However, we would argue that for situations where the NPCs are idle, this might yield a sub-optimal result. This is given by the fact that the decision systems expect the NPC to mostly execute non-repetitive activities, even in the case where the animation would be looped, e.g. during walking, there is still a non-repeating action of transportation from one point to another. This allows for a high degree of control, which is however usually detrimental to the naturalness of the behaviour [4].

Contrary to that, when animating an NPC that is inherently idle, e.g. waiting in a queue, sitting in a class, sleeping, etc., we would like the characters to stay in a loop for a period of time and only occasionally switch to a different action. To this end we present an approach based on discrete time Markov chains (DTMC) which are commonly used for decision making under uncertainty [5]. For our purpose DTMC can be viewed as a direct extension of a state machine, but allows for probabilistic choice of a next state. We show how to use the probabilistic decision making to specify, on average, how long should an NPC stay in a single state, preventing unrealistically fast state switching between possible actions.

Naturally, many other authors considered adding probability to the decision process, e.g. on the level of the state machine [6], [7], the behavior tree [8], by introduction of fuzzy logic [9], etc. These however always only provide a non-deterministic selection to the next selected activity and do not reflect the time of the activity itself. To the best of our knowledge our approach is the only published one that reflects length of an activity directly on the level of decision making process.

II. Methodology

A. Discrete Time Markov Chains

The DTMC [10] is given as a pair $(\mathcal{X}, \mathcal{P})$ where $\mathcal{X} = \{x_1, \ldots, x_n\}$ for $n \in \mathbb{N}^*$ is a set of states and $\mathcal{P} \in \mathbb{P}^{n \times n}$, s.t. $\mathbb{P} = [0, 1]$ is a square transition probability matrix. We denote $p_{i,j}$ the probability of transition from x_i to x_j. The evolution of the systems is then given as a sequence of random variables $X^1, X^2, X^3, \ldots$, that take on values from $\mathcal{X}$. The sequence is said to have the Markov property, meaning that in any state the selection of the next state does not depend on any of the previous states, i.e. $P(X^k = x \mid X^{k-1} = x^{k-1}, \ldots, X^1 = x^1) = P(X^k = x \mid X^{k-1} = x^{k-1})$. Understandably the probabilities of outgoing transitions need to sum to one, i.e. $\forall i \in \{1, \ldots, n\} : \sum_{j \in \{1,\ldots,n\}} p_{i,j} = 1$.

For illustration consider the following system:

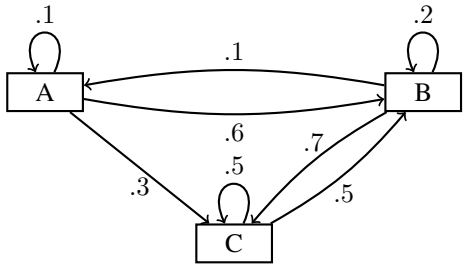
$\mathcal{X} = \{A, B, C\}$

$\mathcal{P} =$

	A	B	C
A	.1	.6	.3
B	.1	.2	.7
C	0	.5	.5

This system can be visualized as a labeled oriented graph:

978-1-5386-4359-4/18/$31.00 ©2018 IEEE



B. Animation State Machine

Obviously, the DTMC can be interpreted as a state machine by an animation engine, e.g. the Unity Mecanim system [11]. Consider for example a character sitting in a classroom who can look at a teacher, out of a window, or on a laptop. We could the assign the states such that:

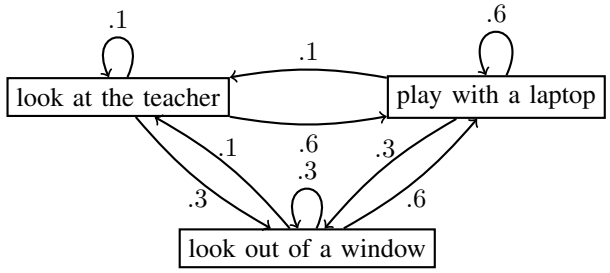
$$A = \text{look at a teacher},$$
$$B = \text{play with a laptop},$$
$$C = \text{look out of a window},$$

we then obtain a state machine whose transitions are guarded by probabilites of taking the transition.

However, in the usual scenario we are less interested in the probability of individual transitions but rather in the probability of a certain action. For example we expect that a student in class spends considerably more time looking at the laptop than focusing on a speaker. We can therefore simplify the construction by assigning probabilities to states directly:

$$P(\text{ look at the teacher }) = .1,$$
$$P(\text{ play with a laptop }) = .6,$$
$$P(\text{ look out of a window }) = .3.$$

This is then interpreted by each incoming transition being assigned the probability of the respective state, i.e.:



We are now in a situation where a character can be driven between its states and we control the probability of the transition happening. However in the case of idle behavior we are usually having shorter loops of singular activity that we would like to keep repeating at least for certain time. In the classroom scenario imagine that each loop is exactly 1 second long, we would then focus on the speaker mostly just for that second as we leave the state with probability of $.3 + .6 = .9$. To prevent this behavior we need to adjust the probabilities on the transitions.

C. Time-adjusted DTMC

First note that we at this point expect each loop to take 1 second. Therefore we need to make sure that if we want to stay for at least t seconds, the probability of the state must be equal to the probability of still being in the state after t random samples.

Lemma 1. *Have $P(x)$ the probability of persisting in s state x for $t \in \mathbb{N}$ consecutive random samples. Then the probability of the a self transition is:*

$$p^t_{x,x} = e^{\frac{\ln(P(x))}{t}}.$$

Proof. We require that the probability $P(x)$ is equal to the probability of $p^t_{x,x}$ being repeated t times in a row, i.e. $(p^t_{x,x})^t$ [10]. Then:

$$\begin{aligned}
(p^t_{x,x})^t &= P(x) \\
\ln((p^t_{x,x})^t) &= \ln(P(x)) \\
t \cdot \ln(p^t_{x,x}) &= \ln(P(x)) \\
\ln(p^t_{x,x}) &= \frac{\ln(P(x))}{t} \\
p^t_{x,x} &= e^{\frac{\ln(P(x))}{t}}
\end{aligned}$$

□

Now we need to set the probabilities of the outgoing transitions. This can be derived from the previous expression in the following way:

Lemma 2. *Have $P(x)$ the probability of persisting in s state x for $t \in \mathbb{N}$ consecutive random samples and a state $y \neq x$ with the probability $P(y)$ of being selected when exiting x. Then the probability of entering y from x is:*

$$p^t_{x,y} = (1 - e^{\frac{\ln(P(x))}{t}}) \cdot \frac{P(y)}{1 - P(x)}.$$

Proof. As the outgoing probability from each state needs to sum to 1 we can see that if the probability of transition to self is $p^t_{x,x}$, then the probability of exiting the state through any of the exit transitions is in total its complement, i.e.:

$$\sum_{y \in \mathcal{X} \setminus x} p^t_{x,y} = 1 - p^t_{x,x}.$$

Now we need to distribute this probability over the outgoing transitions. We know that the sum of probabilities of all the other states but x is also its complement, i.e.:

$$\sum_{y \in \mathcal{X} \setminus x} P(y) = 1 - P(x)$$

For a state $y \neq x$ we therefore know that the under the condition that we exit the state x the probability of entering any $y \in \mathcal{X} \setminus x$ is:

$$P(X^k = y \mid X^{k-1} = x \wedge X^k \neq x) = \frac{P(y)}{1 - P(x)}.$$

Then if we remove the condition $X^k \neq x$ we obtain our final equation:

$$p^t_{x,y} = (1 - e^{\frac{\ln(P(x))}{t}}) \cdot \frac{P(y)}{1 - P(x)}.$$

□

Having derived the transition probability for each state and thus completed the transition matrix it remains to be proven that the transition matrix is sound, i.e. the sum of probabilities out outgoing transitions is 1.

Lemma 3.

$$\forall t \in (0, \infty), \forall x \in \mathcal{X} : \sum_{y \in \mathcal{X}} p^t_{x,y} = 1.$$

Proof. This can be observed already from Lemma 2, however for completion we will derive the proof here:

$$\begin{aligned}
1 &= e^{\frac{\ln(P(x))}{t}} + \sum_{y \in \mathcal{X} \setminus x} ((1 - e^{\frac{\ln(P(x))}{t}}) \cdot \frac{P(y)}{1 - P(x)}) \\
&= e^{\frac{\ln(P(x))}{t}} + (1 - e^{\frac{\ln(P(x))}{t}}) \cdot \sum_{y \in \mathcal{X} \setminus x} (P(y)) \cdot \frac{1}{1 - P(x)} \\
&= e^{\frac{\ln(P(x))}{t}} + (1 - e^{\frac{\ln(P(x))}{t}}) \cdot (1 - P(x)) \cdot \frac{1}{1 - P(x)} \\
&= e^{\frac{\ln(P(x))}{t}} + (1 - e^{\frac{\ln(P(x))}{t}}) \\
&= 1
\end{aligned}$$

□

In this form we can obtain a DTMC that has a specific expectation about how long, on average, each individual state persists. However, up till now we placed an expectation that a single loop animation has the length of exactly 1 second. We now extend the method to arbitrary lengths.

Theorem 1. *Have the states $x, y \in \mathcal{X}$ s.t. $x \neq y$, time $t \in \mathbb{N}$, and $l \in \mathbb{R}^+$ the length of the animation in x. Then the following holds:*

$$\begin{aligned}
p^{t,l}_{x,x} &= e^{\frac{\ln(P(x))}{t \cdot l^{-1}}}, \\
p^{t,l}_{x,y} &= (1 - e^{\frac{\ln(P(x))}{t \cdot l^{-1}}}) \cdot \frac{P(y)}{1 - P(x)}.
\end{aligned}$$

Proof. Follows by replacing t for $t \cdot l^{-1}$ in Lemma 1 and Lemma 2. As we have proven in Lemma 3 that the process is sound for any $t \in (0, \infty)$ and l itself is in $(0, \infty)$ then $t \cdot l^{-1} \in (0, \infty)$ and the above is a simple substitution. □

To illustrate the method we put the requirement on average length to 10 second, i.e. $t = 10$, and consider the length function $L : \mathcal{X} \to \mathbb{R}^+$ that provides the length of the animation in each state and the following valuation of our example:

$$\begin{aligned}
L(\text{ look at a teacher }) &= 10, \\
L(\text{ play with a laptop }) &= 5, \\
L(\text{ look out of a window }) &= 1.
\end{aligned}$$

Then we have, e.g.:

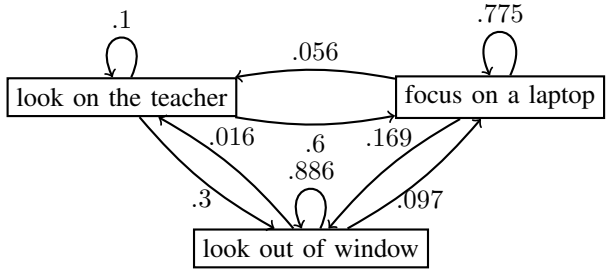
$$\begin{aligned}
p^{10,5}_{B,A} &= (1 - e^{\frac{\ln(.6)}{10 \cdot 5^{-1}}}) \cdot \frac{.1}{1 - .6} \\
&= (1 - e^{\frac{\ln(.6)}{10 \cdot 5^{-1}}}) \cdot .25 \\
&= (1 - e^{\frac{\ln(.6)}{2}}) \cdot .25 \\
&= (1 - \sqrt{e^{ln(.6)}} \cdot .25 \\
&= (1 - \sqrt{.6}) \cdot .25 \\
&\doteq .225 \cdot .25 \\
&\doteq .056
\end{aligned}$$

Note that if the length of the state is equal to the expected time, i.e if $l = t$ then we get:

$$\begin{aligned}
p^{t,l}_{x,y} &= (1 - e^{\frac{\ln(P(x))}{t \cdot l^{-1}}}) \cdot \frac{P(y)}{1 - P(x)} \\
&= (1 - e^{\frac{\ln(P(x))}{1}}) \cdot \frac{P(y)}{1 - P(x)} \\
&= (1 - e^{\ln(P(x))}) \cdot \frac{P(y)}{1 - P(x)} \\
&= (1 - P(x)) \cdot \frac{P(y)}{1 - P(x)} \\
&= P(y)
\end{aligned}$$

meaning the probability remains without any adjustment as we would expect.

The final state with the length-adjusted states then is as follows:



Note that for the states with a shorter length the self-transition probabilities are greatly increased.

III. Conclusion

We have implemented our state machine using the Unity Mecanim system with a set of animations given provided in our running example. An example project is available at [12]. Note that the lengths of individual animations differ from our running example where we selected values better illustrating the mathematical properties of the system.

As our solution can be implemented in just a few lines of code we believe that it presents a useful, novel tool for creation of state machines of non-interactive idle characters, or can be potentially combined with a different NPC control system to take control of the behavior in the time the character is idle.

Additionally, it should be noted that our approach can be easily composited with additional elements typical to realistic

character simulation like eye motion, breathing etc. [13]. Conversely, the approach is not limited to virtual characters only, any animated entity could be modeled in this way.

Lastly, the DTMCs have been very thoroughly studied for their properties. Hence using this well-known framework allows for application of the methods of the field to this use case and further analysis or fine-tuning of this system.

REFERENCES

[1] N. I. Badler, C. B. Phillips, and B. L. Webber, *Simulating humans: computer graphics animation and control*. Oxford University Press, 1993.

[2] J. Gemrot, *Controlling Virtual People*. Univerzita Karlova, Matematicko-fyzikální fakulta, 2017.

[3] G. N. Yannakakis and J. Togelius, *Artificial Intelligence and Games*. Springer, 2017.

[4] H. Van Welbergen, B. J. Van Basten, A. Egges, Z. M. Ruttkay, and M. H. Overmars, "Real time animation of virtual humans: A trade-off between naturalness and control," in *Computer Graphics Forum*, vol. 29, no. 8. Wiley Online Library, 2010, pp. 2530–2554.

[5] O. Alagoz, H. Hsu, A. J. Schaefer, and M. S. Roberts, "Markov decision processes: a tool for sequential decision making under uncertainty," *Medical Decision Making*, vol. 30, no. 4, pp. 474–483, 2010.

[6] K. Perlin and A. Goldberg, "Improv: A system for scripting interactive actors in virtual worlds," in *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. ACM, 1996, pp. 205–216.

[7] L. Chittaro and M. Serra, "Behavioral programming of autonomous characters based on probabilistic automata and personality," *Computer animation and virtual worlds*, vol. 15, no. 3-4, pp. 319–326, 2004.

[8] M. Colledanchise and P. Ögren, "Behavior trees in robotics and AI: an introduction," *CoRR*, vol. abs/1709.00084, 2017. [Online]. Available: http://arxiv.org/abs/1709.00084

[9] P. Leong and M. Chunyan, "Fuzzy cognitive agents in shared virtual worlds," in *2005 International Conference on Cyberworlds (CW'05)*, Nov 2005, pp. 5 pp.–372.

[10] C. M. Grinstead and J. L. Snell, *Introduction to probability*. American Mathematical Soc., 2012.

[11] A. Buckner. (2014) Animate anything with mecanim. [Online]. Available: https://unity3d.com/learn/tutorials/topics/animation/animate-anything-mecanim

[12] A. Streck, "DTMC for control of idle character animation," Jun. 2018. [Online]. Available: https://doi.org/10.5281/zenodo.1290838

[13] A. Shapiro, "Building a character animation system," in *International Conference on Motion in Games*. Springer, 2011, pp. 98–109.

A Machine-Learning Item Recommendation System for Video Games

Paul Bertens, Anna Guitart, Pei Pei Chen and África Periáñez
Yokozuna Data, Silicon Studio
1-21-3 Ebisu Shibuya-ku, Tokyo, Japan
{paul, anna, peipei, africa}@yokozunadata.com

***Abstract*—Video-game players generate huge amounts of data, as everything they do within a game is recorded. In particular, among all the stored actions and behaviors, there is information on the in-game purchases of virtual products. Such information is of critical importance in modern free-to-play titles, where gamers can select or buy a profusion of items during the game in order to progress and fully enjoy their experience. To try to maximize these kind of purchases, one can use a recommendation system so as to present players with items that might be interesting for them. Such systems can better achieve their goal by employing machine learning algorithms that are able to predict the rating of an item or product by a particular user. In this paper we evaluate and compare two of these algorithms, an ensemble-based model (extremely randomized trees) and a deep neural network, both of which are promising candidates for operational video-game recommender engines. Item recommenders can help developers improve the game. But, more importantly, it should be possible to integrate them into the game, so that users automatically get personalized recommendations while playing. The presented models are not only able to meet this challenge, providing accurate predictions of the items that a particular player will find attractive, but also sufficiently fast and robust to be used in operational settings.**

***Index Terms*—recommender systems, ensemble methods, deep learning, online games, user behavior**

I. Introduction

The aim of a recommender system is to provide suggestions to a set of users on items that might be interesting for them. Recommendation systems are commonly found in e-commerce [20], [18] (where users purchase goods like books, clothes or games online), usually implemented through collaborative filtering methods [5]. These work by comparing similar items or similar users based on user ratings. If two users like the same items they are likely similar, and if two items are liked by the same users, those items are probably similar as well. However, as this method does not take into account the contents, new items cannot be recommended. Content-based recommenders can be used to overcome some of these issues by looking at the item in question and finding similarity between items based on inherit properties [24]. A hybrid approach can also be taken, to combine e.g. collaborative information, content features and demographics [11]. A more detailed study into the current limitations and possible extensions of recommendation systems can be found in [1].

The integration of recommendation systems into video games is a relatively new area of research. Previous work has mostly focused on *game* recommendation engines, which present players with suggestions on alternative titles based on the games they have already played [2], [22]. But it is also possible to use recommendation systems to increase player engagement in a game. In modern free-to-play games, users can buy a wide range of virtual items with real money (in-app purchases, IAPs). However, sometimes they can be overwhelmed by the number of items offered and the diversity of playstyles, and this can lead to an increase in the churn rate—as players start to find the contents too difficult and are unable to progress within the game. *Item* recommendation systems can help prevent this problem by offering players a more direct route to the items that could be appealing or useful for them, thereby improving their purchasing and general in-game experience. This may ultimately result into increased revenue [17] by increasing player retention, IAPs and the conversion rate from free to paying users.

To achieve these goals, it is essential to recommend each player the right item—one that fits both their current state and their playing behavior—at the right time. And this is possible because (in contrast to other applications where very limited information is available) every action performed by a player within the game gets recorded. This offers a unique opportunity not only to obtain accurate predictions on the player's in-game behaviour (for example on when and at what level they will leave the game, see [19] and [4]) but also to offer them personalized recommendations of items that are likely relevant to them.

There are previous papers related to item recommendation systems. [23] introduces a recommendation system for the massively multiplayer online first-person shooter game *Destiny*, where players get suggestions on those items that best fit their play style and might improve their performance. They apply similarity measures to global descriptors like total kill count or kill/death ratio. Clusters for the player "base" and "cooldown" stats were derived through k-means clustering, whereas archetypal analysis [7], [21] (which clusters by extreme values rather than centroids [3]) was used to find distinct playstyles. Similar analyses were done for the massively multiplayer online role-playing game *Tera* and the multiplayer strategy game *Battlefield 2: Bad Company 2* [9] or the game *Tomb Raider: Underworld* [8]. In all these cases, players were clustered by their playing behaviour; although no recommendation system was built, behavioral profiling via clustering may be very useful in offering recommendations

978-1-5386-4359-4/18/$31.00 ©2018 IEEE

based on similarity between users.

However, unsupervised clustering methods remains a challenge. In particular, a significant amount of game-specific knowledge, is required to find adequate features that can separate players into the right number of clusters.

A. Aim

While there are several approaches to the problem of developing recommendation systems, here we will explore a different avenue: our aim is to provide a method that predicts the next items a player will purchase, and use this information to recommend them other items. This approach differs from traditional methods as we explicitly use a predictive model.

Such a model allows us to predict, both for new and existing users, the items they are likely to find most appealing based on their playing behaviour. Additionally, it must be robust for operational implementation, to be able to recommend game products automatically, in a variety of game genres, namely different game data distributions.

II. Background

A. Extremely Randomized Trees

Extremely randomized trees (ERTs) [10] extend the randomization of original random forest [13], [6] algorithms by choosing the splitting points randomly instead of computing the ones that are more correlated with the output (which makes random forest an easy biased approach). ERTs are computationally efficient, reducing the variance of the model and preventing overfitting. However the bias can also be larger with this method when the randomization is increased above the optimal level, due to the decrease in the variance.

Breiman implementation of random forest builds an ensemble of decision trees, each of which is fit on a random subset of features [6]. This randomization in the feature selection, combined with the bagging of multiple decision trees, reduces the correlation between trees and increases the overall accuracy of the ensemble.

One of the main advantages of ensemble models is that they are trivially parallelizable, either using multicore processors (as each tree could potentially be trained on a single core) or across multiple machines. This makes them more practical in operational settings, where training and inference have to be completed in a relatively short time, and thus better suited for developing a commercial recommendation system.

B. Deep Neural Networks

Deep neural networks (DNNs) [16] are artificial neural networks with multiple hidden layers. By using nonlinear activation functions (the functions that transform the output at each layer before passing it to the next), DNNs are able to learn highly nonlinear dynamics. Multiple iterations, i.e. epochs, are run to optimize the DNN during the learning process. Rectified linear units (ReLU) are among the most commonly used activation functions nowadays. DNNs that combine ReLU with dropout—a strategy consisting in randomly dropping out some of the units at each layer—have been shown to provide state-of-the-art accuracy in domains such as image classification [15] or speech recognition [12]. Additionally, for sequential data, recurrent neural networks (RNN) or long short-term memory (LSTM) networks [14] have achieved similarly high accuracies in sequence prediction and language modeling.

III. Item Recommendation Model

While RNNs and LSTM networks are able to learn temporal dependencies and eliminate the need for manual feature engineering, they also slow down the training significantly, as they have to learn the relevant features of the time series that lead to an increase in prediction accuracy. On the other hand, by manually calculating general statistics of the time-series data together with other descriptors one can efficiently create a single vector describing the player's behavior and use it in nontemporal models like DNNs or ensemble-based methods such as ERTs.

These are the main challenges related to our approach:

- The model should be able to train and provide inference in production environments scaling to millions of users.
- It should be trainable on mini-batches so that it fits in the memory (ensemble models usually work on the full dataset).
- The time-series data needs to be converted into a single feature vector that accurately represents the player's behavioural patterns (as commented above, tree ensembles and DNNs use static feature vectors, not time series).
- As players make multiple purchases over their lifetime in the game, we must extract their next purchase from multiple time points. Thus the training dataset may become huge if e.g. players remain in the game for several years.

The following sections elaborate on the dataset used and on the way the model was constructed to solve these challenges.

A. Dataset

The data used in our analysis comes from the Japanese card-game *Age Of Ishtaria*, developed by *Silicon Studio*, and contains daily time-series data for each paying user within the period from 2014-09-24 to 2017-05-08 (totaling 33,488 players). It contains information on the number of purchases per item and total sales per item for each user. Players can purchase in-game currency with real money and use it to buy different card-packs (known as *gacha*) containing a random set of cards that can be employed in the game. The data contains 8 different types of items and also has information on e.g. the player's daily level progression, playtime and lifetime.

B. Feature calculation

To convert our time-series into a single static vector we calculate general statistics over the full time-series data for each of the temporal features (e.g. daily playtime or sales). The process is as follows: First we compute the derivative of the time series in order to get its variations (for instance, if we are tracking total level, the derivative gives us the number of level-ups per day). Then we calculate the

mean/variance/skew/kurtosis/maximum over the time series for each of the temporal features. Additionally, to capture behavioral changes of the player between the beginning and end of their lifetime, we also compute the distance for all temporal features over the first and last days in which they logged in. Finally, all these features get concatenated into our final feature vector. By using such a method, the feature calculation can be generalized to any type of temporal data.

C. Sampling to handle multi-label outputs

Players usually make multiple purchases, which means we can have multiple prediction targets (multiple labels) per user. One way of dealing with this is taking some subsample until time t from each player's time series and then find their next purchase after t. This results in a single label we can train on, and allows us to take multiple subsamples to enlarge our training set. Since players could be playing for several years and have hundreds or even thousands of days of playing activity, by using subsampling we can generate different training samples for each player, increasing our effective training dataset and reducing overfitting.

D. Scalability using minibatches

Additionally, the model should be able to scale to millions of players; however, if we generate very large feature vectors (with thousands of features) and sample multiple labels per user, we could end up with datasets with over a billion samples (a thousand samples per user). An efficient way of coping with such huge data sets is to train an ensemble model on subsamples of the total set. Hence, we can train a small subset of trees (~20) on a small sample of a few thousand users and generate the labels directly during training, so that we do not need to store all samples. The final ensemble is formed by combining many such subsets of trees, where each tree was trained on different features, different samples, and different target labels, producing an extremely robust model.

E. Model Specification

1) Output: For each player and item, we generate the probability that they will buy that item on their next purchase day. As the model is trained over all players, once players are in a similar state the model can learn to predict and recommend the right item at the right time for each individual player.

2) Input: We take the full time-series patterns for each user to convert them into a single vector that represents their playing behavior. This conversion is done for all users in a single mini-batch. Multiple mini-batches are generated per epoch (one epoch goes over the entire dataset), and the model is trained on each of these batches.

3) Parameters: The ERT model was trained on subsets of 20 trees for 30 iterations, resulting in a total ensemble size of 600 trees. Each iteration was performed on a subset of ~10k users, which means that a full single epoch was completed after 3 iterations (as the total set has 33,488 players), therefore we had 10 epochs.

For the DNN model, we used two hidden layers of 2048 units and set a dropout probability of 0.5. Additionally, as there were many correlating features, dropout was also applied to the input layer. By randomly dropping some inputs, we reduce overfitting on single features, thereby increasing the robustness of the model. (Recall this was achieved by random subsampling of features in the ERT model.) The network was trained for 30 iterations as well, but each iteration was repeated 5 times, resulting in a total of 50 epochs. Both DNN and ERT are trained on the same data.

IV. Model Evaluation

In order to evaluate the effectiveness of the proposed model, we study the prediction accuracy within an upcoming time window. Predictions are made at a time point t and evaluated at time $t+50$ (where t is measured in days). The training was performed using data up to 2017-03-19, and predictions were verified in the window from 2017-03-20 to 2017-05-08.

Several measures are calculated:

1) *isOnNextPurchaseDate*: Checks whether the predicted item was actually acquired by the player throughout their next purchase day (our training objective).

2) *isNextPurchase*: Checks whether the item that was predicted to be purchased by a certain player was actually acquired by the player on their very next purchase.

3) *isWithinWindow*: Checks whether the predicted item was actually acquired by the player at some point within the time window considered (between $t+1$ and $t+50$).

For all three measures, the accuracy for the top (*predictedMax*), top 2 (*withinTop2*) and top 3 (*withinTop3*) predicted items is calculated, i.e. we check whether the player actually purchased the item that had the highest probability, any of the two items with the two highest probabilities or any of the three items with the three highest probabilities, as per the prediction.

V. Results

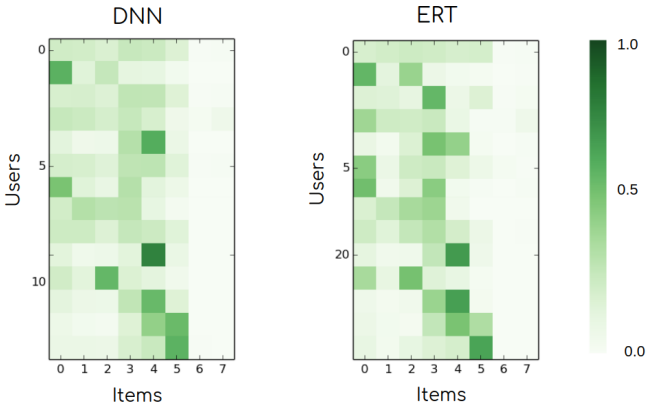


Fig. 1. Predicted probability, for a sample of players and a series of items, that the item will be bought by the player on their next purchase, using the DNN (left) and ERT models (right). (Darker colors correspond to higher probabilities.)

Figure 1 shows the predictions for a subset of users. The DNN (left panel) and ERT (right panel) results exhibit similar patterns (with only slight variations). We see that different

users have different purchase probabilities for each item, which shows that the models are capable of providing personalized predictions for each player based on their playing behaviour.

The accuracy results for both models can be found in Table I. When considering the top 2 and top 3 predictions, both models present similar accuracies, but the ERT is slightly better at identifying the item with the highest probability of being acquired on the next purchase, for all three measures.

TABLE I
ACCURACY RESULTS FOR THE NEXT-PURCHASE PREDICTION IN THE DNN AND ERT MODELS

(DNN)	predictedMax	withinTop2	withinTop3
isOnNextPurchaseDate	44%	68%	81%
isNextPurchase	34%	59%	74%
isWithinWindow	69%	85%	90%
(ERT)	**predictedMax**	**withinTop2**	**withinTop3**
isOnNextPurchaseDate	**47%**	68%	81%
isNextPurchase	**37%**	59%	74%
isWithinWindow	**71%**	85%	91%

VI. DISCUSSION

An item recommendation system for games is essential to provide players with individual rewards or incentives to increase engagement, to maximize in-app purchases and to increase cross-selling and up-selling. We have presented two models to predict which items players will be more attracted to buy in their next purchases. The results show that the predicting performance of the DNN and ERT is similar. However the ERT model yields slightly better results (as shown in Table I) and also scales up more easily in a production environment.

While predictions were made only for a small set of items, the model is trivially extendable to run on hundreds of items, and can be used both for items purchased with real money and for in-game virtual purchases. Future works in this direction will include an evaluation of the recommendation system in terms of total game sales for live video-games.

ACKNOWLEDGEMENTS

We thank Javier Grande for his careful review of the manuscript and Ana Fernández for her support.

REFERENCES

[1] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17(6):734–749, 2005.

[2] S. M. Anwar, T. Shahzad, Z. Sattar, R. Khan, and M. Majid. A game recommender system using collaborative filtering (GAMBIT). In *2017 14th International Bhurban Conference on Applied Sciences and Technology (IBCAST)*, pages 328–332. IEEE, 2017.

[3] C. Bauckhage and R. Sifa. k-maxoids clustering. In *Proceedings of the LWA 2015 Workshops: KDML, FGWM, IR, and FGDB*, pages 133–144, 2015.

[4] P. Bertens, A. Guitart, and Á. Periáñez. Games and big data: A scalable multi-dimensional churn prediction model. In *2017 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 33–36. IEEE, 2017.

[5] J. S. Breese, D. Heckerman, and C. Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, pages 43–52, San Francisco, 1998. Morgan Kaufman.

[6] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[7] A. Cutler and L. Breiman. Archetypal analysis. *Technometrics*, 36(4):338–347, 1994.

[8] A. Drachen, A. Canossa, and G. N. Yannakakis. Player modeling using self-organization in Tomb Raider: Underworld. In *2009 IEEE Symposium on Computational Intelligence and Games (CIG)*, pages 1–8. IEEE, 2009.

[9] A. Drachen, R. Sifa, C. Bauckhage, and C. Thurau. Guns, swords and data: Clustering of player behavior in computer games in the wild. In *2012 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 163–170. IEEE, 2012.

[10] P. Geurts, D. Ernst, and L. Wehenkel. Extremely randomized trees. *Machine learning*, 63(1):3–42, 2006.

[11] C. A. Gomez-Uribe and N. Hunt. The Netflix recommender system: Algorithms, business value, and innovation. *ACM Transactions on Management Information Systems*, 6(4):13, 2016.

[12] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

[13] T. K. Ho. The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(8):832–844, 1998.

[14] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25 (NIPS 2012)*, pages 1097–1105, 2012.

[16] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436, 2015.

[17] V. Lehdonvirta. Virtual item sales as a revenue model: identifying attributes that drive purchase decisions. *Electronic Commerce Research*, 9(1–2):97–113, 2009.

[18] G. Linden, B. Smith, and J. York. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1):76–80, 2003.

[19] Á. Periáñez, A. Saas, A. Guitart, and C. Magne. Churn prediction in mobile social games: towards a complete assessment using survival ensembles. In *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 564–573. IEEE, 2016.

[20] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Analysis of recommendation algorithms for e-commerce. In *Proceedings of the 2nd ACM Conference on Electronic Commerce*, pages 158–167. ACM, 2000.

[21] R. Sifa, C. Bauckhage, and A. Drachen. Archetypal game recommender systems. In *Proceedings of the 16th LWA Workshops: KDML, IR and FGWM*, pages 45–56, 2014.

[22] R. Sifa, A. Drachen, and C. Bauckhage. Large-scale cross-game player behavior analysis on steam. In *Proceedings of the Eleventh AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE-15)*, pages 198–204, 2015.

[23] R. Sifa, E. Pawlakos, K. Zhai, S. Haran, R. Jha, D. Klabjan, and A. Drachen. Controlling the crucible: A novel PvP recommender systems framework for *Destiny*. In *Proceedings of the Australasian Computer Science Week Multiconference, ACSW 2018*, 2018.

[24] A. Van den Oord, S. Dieleman, and B. Schrauwen. Deep content-based music recommendation. In *Advances in neural information processing systems 26 (NIPS 2013)*, pages 2643–2651, 2013.

Learning Battles in ViZDoom via Deep Reinforcement Learning

Kun Shao[†‡], Dongbin Zhao[†‡], Nannan Li[†‡], Yuanheng Zhu[†‡]
[†]The State Key Laboratory of Management and Control for Complex Systems, Institute of Automation, Chinese Academy of Sciences, Beijing 100190, China
[‡]University of Chinese Academy of Sciences, Beijing 100049, China
shaokun2014@ia.ac.cn, dongbin.zhao@ia.ac.cn, linannan2017@ia.ac.cn, yuanheng.zhu@ia.ac.cn

***Abstract*—First-person shooter (FPS) video games play an important role in game artificial intelligence (AI). In this paper, we present an effective deep reinforcement learning (DRL) method to learn battles in ViZDoom. Our approach utilizes the actor-critic with Kronecker-factored trust region (ACKTR), a sample-efficient and computationally inexpensive DRL method. We train our ACKTR agents in two battle scenarios, and compare with the advantage actor-critic (A2C) baseline agent. The experimental results demonstrate that DRL methods successfully teach agents to battle in these scenarios. In addition, the ACKTR agents significantly outperform the A2C agents in terms of all the metrics by a significant margin.**

***Index Terms*—reinforcement learning, deep learning, game AI**

Fig. 1. Game sample of the battle scenario in ViZDoom from the first-person perspective.

I. Introduction

In the last few years, we have witnessed massive progresses of game artificial intelligence (AI) with deep reinforcement learning (DRL) [1] [2]. These DRL agents have achieved impressive performances in various games, including Atari [3], Go [4], Poker [5], and StarCraft [6] [7]. It has been proven that DRL is a general and effective architecture for game AI.

In this paper, we present DRL methods to teach agents to learn battles in ViZDoom [8], which is a first-person perspective 3D environment for visual reinforcement learning research, as shown in Fig. 1. Agents in ViZDoom have many challenges. They have to navigate in high dynamic partially observable maze environment, recognize different objects, and shoot accurately at the enemy targets. All of these behaviors come from high-dimensional image inputs. We apply the state-of-the-art DRL methods to tackle these challenges in two battle scenarios. To evaluate the performance of our agent, we compare the results with the baseline method in terms of some metrics. The organization of the remaining paper is arranged as follows. In Section II, we describe the related work of ViZDoom. Then we present the DRL methods in Section III, and introduce ViZDoom battle scenarios and the learning model in Section IV. In Section V, we present the experimental setup details and the results. Finally, we draw a conclusion of our research.

This work is supported by National Natural Science Foundation of China (NSFC) under Grants No.61573353, No.61603382 and No. 61533017.

II. Related Work

In recent years, deep reinforcement learning has been widely used in game AI, showing superior performance to traditional methods. Here we focus on related work of ViZDoom with DRL methods, especially for battle scenarios.

Kempka *et al.* propose ViZDoom as a novel test-bed for reinforcement learning from visual information. There are various tasks in ViZDoom, and agents have to interact with the 3D world in a first-person perspective. In some simple tasks, e.g., the basic and the health gathering, the deep Q-Network (DQN) agent achieves acceptable performances [8]. Kulkarni *et al.* use the successor representations to decompose the value function into a reward predictor and a successor map, and generalize it in the end-to-end DRL architecture. This method has the increased sensitivity to distal reward changes and the ability to extract bottleneck states in ViZDoom [9]. Lample *et al.* present a method to augment the deep recurrent Q-network (DRQN) model to exploit game feature information during training, which improves the training speed and performance dramatically. This model bases on the proposed action-navigation architecture, and outperforms built-in agents as well as average humans in the deathmatch battle scenarios [10]. The agent, named Arnold, wins the first place for track 2 in the ViZDoom AI competition 2017. Tian *et al.* propose a framework that combines the asynchronous advantage actor-critic (A3C) model and curriculum learning [11]. This agent learns to move and shoot via playing against built-in agents in a progressive manner and wins the first place for track

1 in the ViZDoom AI competition 2016. Dosoviskiy *et al.* propose the direct future prediction (DFP), which utilizes a high-dimensional sensory stream and a lower-dimensional measurement stream as the inputs to provide a rich supervisory signal, and trains a sensorimotor control model by interacting with the environment [12]. DFP successfully generalizes across environments and goals, and outperforms state-of-the-art DRL methods on challenging deathmatch tasks. This agent wins the first place for track 2 in the ViZDoom AI competition 2016.

III. Deep Reinforcement Learning

A. Reinforcement Learning and Actor-Critic Methods

In the reinforcement learning paradigm, an agent learns by trial and error, and determines the behavior from its own experiences with the environment [13]. Here we consider an agent interacting with a discounted Markov decision process (MDP) (S, A, γ, P, r). At time t and state s_t, the agent chooses an action a_t according to the policy $\pi(a_t|s_t)$. After receiving the action, the environment produces a reward r_{t+1} and transitions to the next state s_{t+1} according to the transition probability $P(s_{t+1}|s_t, a_t)$. The process continues until the agent reaches a terminal state. The goal of the agent is to maximize the expected discounted cumulative rewards under the policy π with discount factor $\gamma \in (0, 1]$

$$\mathbb{E}_\pi[R_t] = \mathbb{E}_\pi[\sum_{i=0}^{\infty} \gamma^i r_{t+i}]. \tag{1}$$

Policy gradient methods parameterize the policy $\pi_\theta(a_t|s_t)$ directly and update parameter θ to maximize the objective function $J(\theta)$. In its general form, $J(\theta)$ is defined as

$$J(\theta) = \mathbb{E}_\pi[\sum_{t=0}^{\infty} \Psi_t \log \pi_\theta(a_t|s_t)]. \tag{2}$$

Based on policy gradient methods, actor-critic reinforcement learning methods use a value function to approximate Ψ_t. Mnih *et al.* propose the asynchronous advantage actor-critic method, which asynchronously executes multiple agents on multiple instances of the environment [14]. With multiple agents playing concurrently and optimizing the networks through asynchronous gradient descent, A3C decorrelates the training data into a more stationary process and improves the performances of a number of tasks.

As the synchronous version of A3C, A2C optimizes the learning model synchronously and uses the GPU to accelerate learning process [15]. In A2C, the objective function is defined as

$$R'_t = \sum_{i=0}^{k-1} \gamma^i r_{t+i} + \gamma^k V_{\theta_v}(s_{t+k}), \tag{3a}$$

$$A_{\theta,\theta_v}(s_t, a_t) = R'_t - V_{\theta_v}(s_t), \tag{3b}$$

$$J(\theta) = \mathbb{E}_\pi[\sum_{t=0}^{\infty} A_{\theta,\theta_v}(s_t, a_t) \log \pi_\theta(a_t|s_t) + \beta H_\theta(\pi(s_t))]. \tag{3c}$$

θ_v are the parameters of the value network, and $H_\theta(\pi(s_t))$ is an entropy term used to encourage exploration during the training process.

B. Actor Critic using Kronecker-Factored Trust Region

A2C and other DRL methods are usually trained using simple variants of stochastic gradient descent (SGD), which are inefficient first-order methods. Natural gradient descent can perform gradient updates efficiently, which follows the steepest descent direction and uses the Fisher metric as the underlying metric. More recently, Kronecker-factored approximated curvature (K-FAC) is proposed as a scalable approximation to natural gradient, which can be used to the Fisher matrix to perform approximate natural gradient updates efficiently. Wu *et al.* combine the actor-critic, trust-region policy optimization (TRPO) and K-FAC, and introduce the actor-critic using Kronecker-factored trust region (ACKTR) [16]. ACKTR extends the framework of natural policy gradient and optimizes both the actor and the critic using K-FAC. For the actor, we use the policy distribution to define the Fisher matrix

$$F = \mathbb{E}_{p(\tau)}[\nabla \log \pi(a_t|s_t) \nabla \log \pi(a_t|s_t)^T]. \tag{4}$$

For the critic, the output of the critic v is defined to be a Gaussian distribution, and we define the Fisher matrix with respect to this Gaussian output distribution.

$$p(v|s_t) \sim \mathcal{N}(v; V(s_t), \sigma^2) \tag{5}$$

When actor and critic share lower-layer representations, we define the joint distribution as $p(a, v|s)$, and apply K-FAC to approximate the Fisher matrix to perform updates simultaneously.

$$p(a, v|s) = \pi(a|s)p(v|s) \tag{6a}$$

$$F = \mathbb{E}_{p(\tau)}[\nabla \log p(a, v|s) \nabla \log p(a, v|s)^T] \tag{6b}$$

Moreover, ACKTR adopts the trust region formulation of K-FAC, choosing the effective step size η. The natural gradient is performed with the following updates.

$$\eta = min(\eta_{max}, \sqrt{\frac{2\delta}{\Delta\theta^{\mathbf{T}} \hat{F} \Delta\theta}}) \tag{7a}$$

$$\theta \leftarrow \theta - \eta F^{-1} \nabla_\theta L \tag{7b}$$

where η_{max} is the learning rate and δ is the trust region radius.

ACKTR is the first scalable trust region natural gradient method for actor-critic DRL, and improves the sample efficiency of current methods significantly. In the following experiments, we will use this method to train our agents.

IV. Learning Model for Battles in ViZDoom

A. Battles in ViZDoom

We consider two battle scenarios in ViZDoom as the testbeds for our deep reinforcement learning agents, as shown in Fig. 2. In each scenario, the agent is armed and is under attack by enemies in a maze. The enemies spawn abundantly, move around in the environment, and shoot at our agent. In the second task, apart from enemies, health kits and ammunition

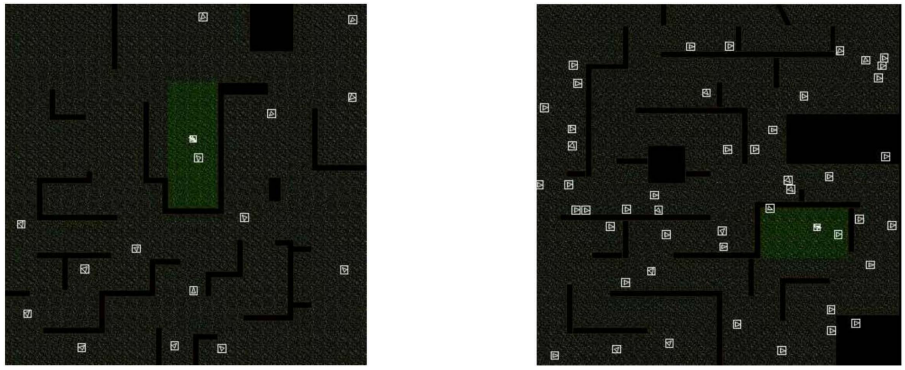


Fig. 2. Examples of the ViZDoom battle maps in the experiment, left: the map of the first battle; right: the map of the second battle.

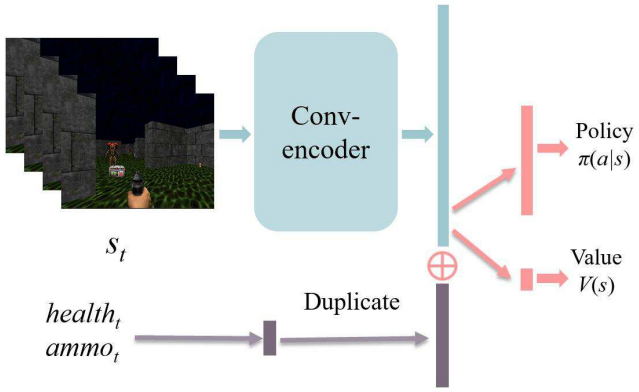


Fig. 3. The architecture of the actor-critic DRL model for battle scenarios in ViZDoom. The model takes in recent four frames visual inputs s_t, game variables $health_t$, $ammo_t$, and produces policy $\pi(a|s)$, value function $V(s)$.

are sporadically distributed throughout the environment and can be collected by the agent.

We will train the agent to learn battles with complex visual inputs in these two scenarios. The agent has seven actions: move forward, move backward, move left, move right, turn left, turn right, and attack. An episode comes to an end when the agent is dead or reaches the maximum game steps. The measurements that the agent can access in these scenarios are the health point, the ammo, and the kill count. The objective of the agent is to kill as many enemies as possible.

We use the reward shaping method in these scenarios. Basically, when the agent is dead, it will receive a penalty of -100. Furthermore, to accelerate the learning process, we give additional rewards for gathering a health kit and a ammo kit, which are set to +10. If our agent is attacked by the enemies, it will receive a penalty of -10. And we add a penalty for wasting ammo, which is set to -2. In the training process, we reshape the total reward and divide it by 100. Each episode finishes after 4200 steps.

B. Learning model for Battles

We follow the architecture of the Facebook F1 bot that uses the actor-critic deep reinforcement learning model to tackle the battle scenarios in ViZDoom. The actor-critic method used in our experiment learns both a policy $\pi_\theta(a_t|s_t)$ and a value function $V_{\theta_V}(s_t)$. The model receives a state observation s_t, game variables $health_t$, $ammo_t$, and uses deep convolutional neural networks as the encoder. The resolution of the original grey-scale visual image is 160×120. We stack four recent observations as the inputs to track historical traces of the agent. In the data preprocessing, we resize the visual observation to 84×84.

The neural network used in the experiment follows a similar architecture in the DQN paper. The network uses a convolutional layer with 32 filters of size 8×8 with stride 4, followed by a convolutional layer with 64 filters of size 4×4 with stride 2, followed by a convolutional layer with 64 filters of size 3×3 with stride 1, followed by a fully connected layer with 512 hidden units. All four hidden layers are followed by a rectifier nonlinearity. Thereafter, we concatenate the game variables tensor and use the softmax function to output the policy $\pi_\theta(a_t|s_t)$, and use one linear output for the value function $V_{\theta_V}(s_t)$. We share all the non-output layers in the actor network and the critic network. This layer-sharing mechanism can stabilize the training process. To balance exploration and exploitation in reinforcement learning, we use the Boltzmann exploration method to select actions. The details of the actor-critic DRL model for ViZDoom battle scenarios are depicted in Fig. 3.

V. Experiments

A. Experimental Setup

The experiments are performed on the two presented ViZDoom battle scenarios with the following setup. We refer to the hyperparameters of DRL model in the OpenAI baseline [17]. In each experiment, we open 128 processes running on a single machine with one Nvidia P100 GPU. The DRL methods perform updates after every 20 action steps. The discount factor is set to 0.99. We use the standard non-centered RMSProp optimizer to update the A2C agent, and the K-FAC optimizer to update the ACKTR agent.

B. Experimental Results

We present the experimental results of our ACKTR agent and the A2C agent in Fig. 4. In both battle scenarios, the

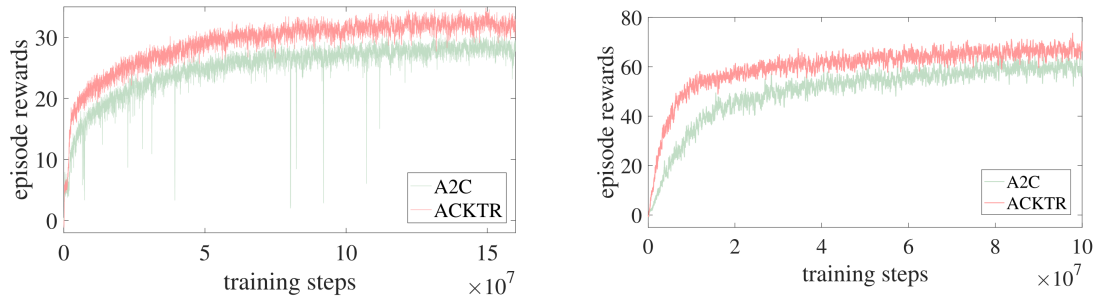


Fig. 4. Episode rewards achieved by DRL agents in the two battle scenarios during training. These rewards are averaged among 128 processes.

TABLE I
TEST RESULTS OF DRL AGENTS IN TWO BATTLE SCENARIOS.

Battles	Agents	Rewards	LivingSteps	KillCounts
Battle-1	A2C	27.8±6.9	1287.7 ±530.1	23.3±5.6
Battle-1	ACKTR	**32.1** ± 7.2	**1303.5**±514.2	**26.2**±5.7
Battle-2	A2C	60.7±19.8	3758.5.7±890.8	20±7.6
Battle-2	ACKTR	**66.1**±21.6	**3975.7**±580.1	**24**±6.6

curves of episode rewards increase very fast in the early stage, and the models converge at the end of training. In the first battle scenario, we train the agents for 150 million steps. In the second battle scenario, the training process ends after 100 million steps.

After training, we test the agents in each battle scenario for 100 games. We record the episode rewards, the living steps and the killcount, which are presented in Table I. From the killcount results, we can see that both DRL agents can successfully learn how to battle in these scenarios. Compared with the A2C agent, the ACKTR agent has better performances in terms of all the metrics. In particular, the ACKTR agent achieves 12.4% and 20% higher killcount scores than the A2C agent in the two battles. Moreover, the ACKTR agent significantly outperforms the A2C agent in terms of sample efficiency by a significant margin. And the performance of ACKTR agent is more stable than the A2C agent, since the A2C agent has several obvious declines during training.

VI. CONCLUSION AND FUTURE WORK

In this paper, we present the efficient ACKTR deep reinforcement learning method to learn battles in ViZDoom. The experimental results show that the DRL agents can successfully tackle these battle scenarios after training. The ACKTR agent significantly outperforms the A2C agent in terms of all the metrics. In the future, we will introduce auxiliary tasks to improve the performance and generalization of our model. And we will train an agent to play the full deathmatch scenario with DRL methods.

REFERENCES

[1] D. Zhao, K. Shao, Y. Zhu, D. Li, Y. Chen, H. Wang, D. Liu, T. Zhou, and C. Wang, "Review of deep reinforcement learning and discussions on the development of computer Go," *Control Theory and Applications*, vol. 33, no. 6, pp. 701–717, 2016.

[2] Z. Tang, K. Shao, D. Zhao, and Y. Zhu, "Recent progress of deep reinforcement learning: from AlphaGo to AlphaGo Zero," *Control Theory and Applications*, vol. 34, no. 12, pp. 1529–1546, 2017.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, and G. Ostrovski, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[4] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, d. D. G. Van, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, and M. Lanctot, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[5] M. Moravik, M. Schmid, N. Burch, V. Lisy, D. Morrill, N. Bard, T. Davis, K. Waugh, M. Johanson, and M. Bowling, "Deepstack: Expert-level artificial intelligence in heads-up no-limit poker," *Science*, vol. 356, no. 6337, pp. 508–513, 2017.

[6] O. Vinyals, T. Ewalds, S. Bartunov, P. Georgiev, A. S. Vezhnevets, M. Yeo, A. Makhzani, H. Kttler, J. Agapiou, and J. Schrittwieser, "StarCraft II: A new challenge for reinforcement learning," *arXiv preprint arXiv:1708.04782*, 2017.

[7] K. Shao, Y. Zhu, and D. Zhao, "StarCraft micromanagement with reinforcement learning and curriculum transfer learning," *IEEE Transactions on Emerging Topics in Computational Intelligence*, DOI:10.1109/TETCI.2018.2823329, 2018.

[8] M. Kempka, M. Wydmuch, G. Runc, J. Toczek, and W. Jakowski, "ViZDoom: A Doom-based AI research platform for visual reinforcement learning," in *IEEE Conference on Computational Intelligence and Games (CIG)*, 2017, pp. 1–8.

[9] T. D. Kulkarni, A. Saeedi, S. Gautam, and S. Gershman, "Deep successor reinforcement learning," *arXiv preprint arXiv:1606.02396*, 2016.

[10] G. Lample and D. S. Chaplot, "Playing FPS games with deep reinforcement learning," in *AAAI Conference on Artificial Intelligence (AAAI)*, 2016, pp. 2140–2146.

[11] Y. Wu and Y. Tian, "Training agent for first-person shooter game with actor-critic curriculum learning," in *International Conference on Learning Representations (ICLR)*, 2017.

[12] A. Dosovitskiy and V. Koltun, "Learning to act by predicting the future," in *International Conference on Learning Representations (ICLR)*, 2017.

[13] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[14] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International Conference on Machine Learning (ICML)*, 2016, pp. 1928–1937.

[15] C. Alfredo, C. Humberto, and C. Arjun, "Efficient parallel methods for deep reinforcement learning," in *The Multi-disciplinary Conference on Reinforcement Learning and Decision Making (RLDM)*, 2017, pp. 1–6.

[16] W. Yuhuai, M. Elman, L. Shun, G. Roger, and J. Ba, "Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation," in *Advances in Neural Information Processing Systems (NIPS)*, 2017, pp. 1285–5294.

[17] P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, "Openai baselines," https://github.com/openai/baselines, 2017.

Anxious Learning in Real-Time Heuristic Search

Vadim Bulitko
Department of Computing Science
University of Alberta
bulitko@ualberta.ca

Kacy Doucet
Department of Computing Science
University of Alberta
kjdoucet@ualberta.ca

***Abstract*—Real-time heuristic search methods are used when planning time available per agent's move is severely limited (e.g., while pathfinding in a video game). Such agents interleave planning and plan execution. As the agent has to move before a complete plan is computed, it is prone to be misguided by inaccuracies in its heuristic. To get out of heuristic depressions, such agents update their heuristic over time. The usual update process requires multiple state revisits which can make the agent appear irrational to the player. To alleviate such map "scrubbing" we propose a new learning mechanism inspired by the psychological notion of anxiety. Our agent maintains a level of anxiety which increases due to state revisits and decays naturally over time. Agent's anxiety causes it to update the heuristic more aggressively thus filling heuristic depressions quicker. Such anxiety-accelerated learning can be used on top of other real-time heuristic search techniques. Empirical evaluation on video-game pathfinding benchmarks demonstrates benefits for the average solution quality when the new mechanism is used by itself or in combination with expendable state marking.**

***Index Terms*—artificial intelligence, heuristic search**

I. Introduction

Artificial Intelligence (AI) in video games is often used to control non-playable characters (NPCs) in real-time, as the game progresses. Such AI needs to be responsive to the player's actions as well as other events in the game. In the context of NPC pathfinding, this means that often an NPC move is due before a complete path can be computed. Real-time heuristic search (RTHS) algorithms [1] are specifically designed for such problems, interleaving planning and plan execution by computing only a few next moves. Such reactivity and locality [2] comes at the cost of suboptimal actions – the myopic nature of their planning can easily guide the agent into a deadend. To get out of such traps an RTHS agent updates its estimates of the distance to the goal (called *heuristic*). Traditionally, learning rules based on the Bellman optimality equation are used. They keep the heuristic locally consistent by keeping updates small. Such conservative learning can result in state revisits [3], [4] which look bad in a video game.

Various mechanisms have been suggested to alleviate such map "scrubbing". In this paper we consider a recent technique that reduces the search space by pruning some states while preserving graph connectivity [5]. This technique has been found powerful in practice [6] but is limited by the fact that the pruning is permanent and thus states that critically connect regions of the map can never be pruned. Suppose an agent wanders into a maze and gets lost there due to an inaccurate heuristic (a la [7]). Should the agent prune the entrance to the maze to prevent re-entry? What if going through the maze is the only way to get to the goal? Pruning is binary and does not afford the agent a way to keep the state in the map but remember its bad experience in it.

We propose an alternative to pruning states. Instead of removing them from the map entirely, we have the agent merely make them less attractive by aggressively increasing their heuristic. To do so we introduce the notion of *anxious learning* which increases the heuristic of the agent's current state more rapidly than an update rule based on the Bellman optimality equation would. The extra learning amount is controlled by the agent's current level of anxiety. In psychology, anxiety is defined as a state of heightened awareness signalling a potential threat or negative outcome [8]. An agent's revisiting states ("walking in circles") indicates inaccuracies in the heuristic and the possibility of costly detours, so we increase the agent's anxiety level when this happens. The anxiety level naturally decays over time.

The new mechanism is the primary contribution of this paper. We evaluate its effectiveness on top of a basic RTHS algorithm as well as in combination with marking expendable states. We find that the two techniques are competitive but their combination outperforms either one of them individually.

II. Problem Formulation

In this paper we are tackling the standard real-time heuristic search problem [9]. The agent is traversing an undirected search graph $G = (S, E)$ comprised of a finite set of vertices/states S connected by a finite set of edges $E \subset S \times S$. Each edge is weighted by a cost function $c : E \to \mathbb{R}$. All costs are positive. There are no self-loops in the graph. Time proceeds in discrete steps. At time t the agent occupies a single state s_t which it changes to a neighboring state $s_{t+1} \in N(s_t)$ by traversing the edge $(s_t, s_{t+1}) \in E$. Neighbors of state s are denoted by $N(s)$. The agent starts in a start state s_0 and at time T arrives in the goal state s_g ($s_0 \neq s_g$). The agent's solution is the path $(s_0, s_1, \ldots, s_g)$. The cumulative cost of all edges in that path is the *solution cost*. The *solution suboptimality* is then the ratio of the solution cost to the smallest possible solution cost. For instance, solution suboptimality of 1 indicates an optimal solution while solution suboptimality of 2 indicates that the path found is twice as long as needed.

In real-time heuristic search, solution suboptimality on a single problem is rarely of interest. Instead, an algorithm is evaluated in terms of its average performance over a set of

978-1-5386-4359-4/18/$31.00 ©2018 IEEE

benchmark problems. The problems differ from each other in terms of the search graph and/or the start and goal states. Solution suboptimality has a long-tailed distribution with very large suboptimality on a few problems. The usual arithmetic mean is sensitive to such high values. Consequently, in this paper we will primarily use the geometric mean defined as $\sqrt[n]{\prod x_i}$ to be less sensitive to such data points.* To indicate variance in the data, we will use geometric standard deviation factor y which is multiplicative relative to the geometric mean x, denoted by $x \times/\div y$.

In its search, the agent has access to the entire graph as well as a *heuristic function* $h : S \to \mathbb{R}$ which estimates the remaining cumulative distance between a state and the goal. The search is, however, real-time and the agent is allowed to consider only up to M states in the graph before it is required to traverse an edge and M is independent of the graph size $|S|$. Since the agent is required to act before it can compute a complete solution, state revisits and backtracking are likely. To avoid looping forever, the agent updates its heuristic as it traverses the graph. The initial, and usually inaccurate, heuristic is given to the agent as an input.

III. Related Work

Computational models of emotions have long been used in the design of NPC agents regarding both their appearance [10] and their actions [11]. More recently researchers applied a computational model of emotions to pathfinding [12]. In doing so the agent felt a positive valence when the goal was within its line of sight and it was moving farther away from the start state. The positive valence would then be used to increase the agent's rewards in the context of reinforcement learning. However, the results show learning performance comparable to that of valence-free learning. Other researchers have injected internal rewards based on flow [13] and learning progress [14] into the stream of external rewards.

Even more recent work used RTHS as a vehicle to study unscripted and seemingly irrational behavior in which an agent avoids areas of the map where it had bad experience [7]. To do so, the authors replaced the agent's usual heuristic with the agent's solution cost if it were to reach the goal from each state using the basic heuristic. They discovered that such knowledge of one's own behavior reliably led to worse results as the agent would avoid areas of the map not because they were objectively bad (i.e., far from the goal) but because the agent would get lost there due to inaccuracies in its heuristic. While curious, the work used an impractical heuristic. We realize the idea of anxiety in a practical fashion which also happens to improve solution suboptimality.

IV. Our Approach

Intuition. The Learning Real-Time A* (LRTA*) algorithm [1] used a learning rule based on the Bellman optimality equation (called *minimin*) to make the heuristic of the current state consistent with that of neighboring states. While this allows for convergence to an optimal solution over repeated trials, the updates to the heuristic function are necessarily low and force the agent to revisit states repeatedly before it can escape a heuristic depression (known as scrubbing) [4].

We use a more aggressive learning rule to fill the heuristic depressions faster. Unlike prior work which uniformly accelerated learning [15], [16], our learning acceleration is dynamic and only activated by state revisits. Other work has used expendable states a way of preventing state revisits and filling heuristic depressions [5], but when heuristic depressions have obstacles in them, marking expendable states results in maze-like structures whose heuristic still needs to be updated as the agent moved around the maze. Anxious learning improves upon this by reducing scrubbing and allowing the agent to escape such mazes faster.

Algorithmic Details. Our approach to anxious learning is presented as Algorithm 1. The agent starts with no anxiety in line 3. As the agent traverses the graph (line 4)† it tracks its state revisits via the function L. When in the state s_t, the loop distance $L(s_t)$ is defined as the cost of all edges traversed by the agent since its previous visit to s_t.‡ The loop distance relative to the agent's current estimate of the remaining distance to goal (i.e., $^{L(s_t)}/_{h_t(s_t)}$) is then compared against the *memory span* $L_{\max}$ and it is reset to 0 if the loop is too long (line 6). The agent's anxiety level a is increased proportionally to the relative loop length in line 8 with the *anxiety rate* α. It is reduced by the *anxiety decay* Δ_a in line 7. Anxious learning then happens in line 9 where the agent additionally increases its heuristic by a.

If a state is expendable (i.e., $\mathcal{E}(s_t)$ holds) and its heuristic has been increased then a state is removed from the graph in line 11. The expendable mechanism [5], [9] is controlled by a switch E. Note that while in the original formulation, the heuristic increase requirement ($h_{t+1}(s_t) > h_t(s_t)$) meant that states are expended only around pathfinding obstacles, our agent can increase the heuristic even in open areas if the agent is anxious ($a > 0$) so states can be expended in the open too.

Finally the agent selects the neighbor greedily according to its updated heuristic in line 12 and the cycle repeats.

V. Empirical Evaluation

We have an empirical evaluation setup similar to recent work [9] and use their set of 493298 pathfinding problems from MovingAI benchmarks [17]. The problems were distributed over 342 video-game maps found in the repository and originating from the video games *StarCraft*, *WarCraft III*, *Baldurs Gate II* (maps scaled up to 512×512 cells) and *Dragon Age: Origins*. The maps were treated as 8-connected grids. Each grid cell is a vertex/state in the search graph. Edges between cardinally (diagonally) neighboring cells cost 1 ($\sqrt{2}$) to traverse. The initial heuristic is octile distance which ignores walls but otherwise is accurate.

*For log-normally distributed data geometric mean is equal to median.

†We exit the **while** loop when the travel cost accumulated so far makes the solution suboptimality exceeds *an priori* specified cutoff.

‡$L(s_t) = 0$ if it is the agent's first visit to s_t.

Algorithm 1: Anxious Real-time Heuristic Search

input : search problem (G, c, s_0, s_g, h); anxiety control parameters α, Δ_a, $L_{\max}$; expendable pruning switch E
output: path $(s_0, s_1, \ldots, s_T), s_T = s_g$

$t \leftarrow 0$
$h_t \leftarrow h$
$a \leftarrow 0$
while $s_t \neq s_g$ **do**
  if $L(s_t)/h_t(s_t) \geq L_{\max}$ **then**
    $L(s_t) \leftarrow 0$
  $a \leftarrow \max\{0, a - \Delta_a\}$
  $a \leftarrow a + \alpha \cdot L(s_t)/h_t(s_t)$
  $h_{t+1}(s_t) \leftarrow \max\left\{h_t(s_t), a + \min_{s \in N(s_t)} (c(s_t, s) + h_t(s))\right\}$
  if E & $h_{t+1}(s_t) > h_t(s_t)$ & $\mathcal{E}(s_t)$ **then**
    $G \leftarrow G \setminus \{s_t\}$
  $s_{t+1} \leftarrow \arg\min_{s \in N(s_t)} (c(s_t, s) + h_t(s))$
  $t \leftarrow t + 1$
$T \leftarrow t$

Many problems in the set were fairly easy even for basic LRTA* (e.g., if the start and goal states are within the same open space on a map, the initial heuristic offers perfect guidance). Averaging suboptimality over such problems reduces the distinction between competing algorithms. Thus, we excluded all problems which basic LRTA* solved with suboptimality below 5, leaving 282064 problems (i.e., approximately 57%) in the set. To have a smaller set, we used the same removal technique on a subset of the original scenario which originally contained 1000 problems selected from the 493298 problems. The removal procedure left 563 problems in the set.

Effects of Anxiety. Our anxious learning mechanism is controlled by three parameters as described above. We conducted a sweep of the parameter space to determine the most suitable combination. We tried all combinations of the anxiety rate $\alpha \in \{1, 50, 100, 500, 2000, 3000, 4000, 5000, 10000\}$, anxiety decay $\Delta_a \in \{0, 1, 3, 5, 10, 15, 20\}$ and the memory span $L_{\max} \in \{0.005, 0.07, 0.13, 0.19, 0.25, 0.31, 0.38, 0.44, 0.5\}$. For each of the 567 parameter combinations, we ran the resulting algorithm twice (once with expendable-state marking turned on and once with it turned off) on each problem from the 563-problem set described above. Among the algorithms that could solve all problems in the set under the 10^4 suboptimality cutoff, the best geometric mean suboptimality was achieved with the anxiety rate of 5000, anxiety decay of 15 and the memory span of 0.44 (when expendable-state marking was off). With expendable-state marking on, the best anxiety parameters found were $5000, 10, 0.07$ respectively.

Table I compares the two resulting algorithms with the baseline LRTA* as well as LRTA* with expendable-state marking, both of which do not have any anxiety. The cutoff was set to 10^5; all algorithms solved all problems. Marking expendable states by itself brings a substantial boost to the LRTA* performance. Anxiety by itself yields a better geometric mean yet. Finally, the combination of expendable states and anxiety yields the best performance. LRTA* with anxiety but without marking expendable states has a tremendous per move anxiety, adding approximately 1.18×10^6 to its heuristic on an average step. With expendable states enabled, the per-step anxiety add-on drops to approximately 4.6×10^3. This difference is largely due to the different memory span values 0.07 versus 0.44. Furthermore, expended states are removed from the map and cannot be revisited thereby affecting the evolution of the agent's anxiety level.

TABLE I
ALGORITHMS COMPARED ON THE 563-PROBLEM SET.

Base	E	Anxiety	Suboptimality	Anxiety/move
LRTA*	off	$0, 0, 0$	$171.87\,\times/\div\,7.05$	0
LRTA*	on	$0, 0, 0$	$19.31\,\times/\div\,4.30$	0
LRTA*	off	$5000, 15, 0.44$	$17.35\,\times/\div\,4.64$	1180028.6
LRTA*	on	$5000, 10, 0.07$	$13.03\,\times/\div\,3.27$	4622.9

Portability of Anxiety Parameters. We then tested the anxiety parameters tuned for the 563-problem set on the set of 282064 problems. We substituted basic LRTA* with a previously published algorithm known for high performance on MovingAI problem sets [9]. The algorithm is denoted by $2.654 \cdot \text{median}_{0.153}(2.486 \cdot c + h)$+da+E and uses additional algorithmic blocks of depression avoidance (da) [18], non-minimum learning operator and learning weights [9], [15], [16]. The results are found in Table II with the previously published algorithm denoted by "[9]". The suboptimality cutoff was 10^6; all algorithms solved all problems.

TABLE II
ALGORITHMS COMPARED ON THE 282064-PROBLEM SET.

Base	E	Anxiety	Suboptimality	Anxiety/move
[9]	on	$0, 0, 0$	$14.99\,\times/\div\,3.45$	0
LRTA*	on	$0, 0, 0$	$18.76\,\times/\div\,4.22$	0
LRTA*	off	$5000, 15, 0.44$	$19.56\,\times/\div\,5.07$	1421233.7
LRTA*	on	$5000, 10, 0.07$	$13.52\,\times/\div\,3.41$	6067.0

The combination of anxiety and marking expendable states once again wins against either of them individually. More surprisingly, LRTA* with anxiety and marking expendable states outperforms even an algorithm that uses other building blocks such as weighted learning and depression avoidance.

When to Be Anxious. Results in the previous section indicate that using anxious learning on top of expendable state marking is beneficial overall. There are, however, problems where anxious learning makes the solution worse. If one knew which problems to use anxiety on, further performance gains could be had. For instance, on the 563-problem set, using anxiety only on problems where it helps reduces mean suboptimality from $13.03\,\times/\div\,3.27$ to $11.83\,\times/\div\,3.01$.

To analyze when anxiety is beneficial, we ran LRTA*+E (second row in Table II) and LRTA*+E with anxiety (last row in Table II) on each of the 282064 problems. We defined the *benefit of anxiety* as the difference of the solution suboptimality without anxiety and the solution suboptimality with

anxiety. We computed the (arithmetic) mean and (arithmetic) standard deviation of the benefit and removed 189 problems whose benefit was more than 2 standard deviations away from the mean as outliers. We then clustered the remaining benefit values into $k = 3$ clusters using k-means.We ran the clustering process three times and obtained the same three clusters. The largest cluster contained 268060 problems and the (arithmetic) mean benefit of using anxiety was 11.8. Cluster 2 contained 12032 problems and the (arithmetic) mean benefit was 383.4. The final cluster contained 1783 problems and the (arithmetic) mean benefit was -546.1 (i.e., anxiety degraded solution suboptimality on average). We discarded the largest cluster (where anxiety had least effect on average) and considered the two clusters where anxiety effects were more pronounced. We then visualized each problem from each cluster as an image, similar to recent work [19], [20]. Each open grid cell became a single white pixel. Wall cells were black pixels. The agent's start/goal states became a 9×9–pixel red/green square centered in the start/goal cell (Figure 1).

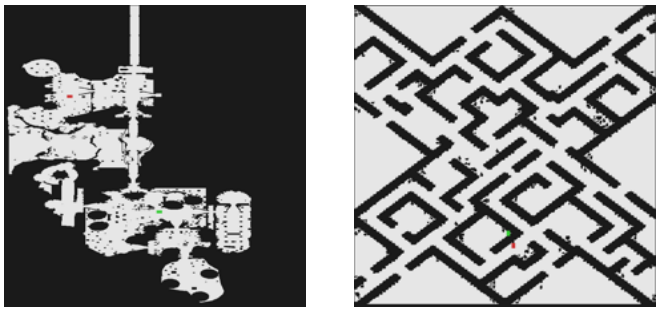


Fig. 1. Two sample problems: where anxiety on top of LRTA*+E is helpful (left) and where anxiety is detrimental (right).

We then trained AlexNet [21] on the images in the two clusters and used it to turn anxiety on and off on a per-problem basis. While the test binary classification accuracy reached 87.5%, the resulting suboptimality was slightly worse than leaving anxiety always on with LRTA*+E.

VI. Current Shortcomings and Future Work

Similar to recent work on automated RTHS algorithm selection we investigated only a subclass of algorithms with the lookahead depth of one. Future work will generalize our definition of anxious learning on algorithms with deeper lookahead such as LSS LRTA* [22] or PALMA [6]. It will also be of interest to compare weighted learning [15], [16] and lateral learning [23] to our anxiety mechanism. The latter has a potential advantage by being activated only when the agent revisits states. It will also be of interest to see if the number of revisits (i.e., the degree of scrubbing) is a useful input to the anxiety level of the agent.

It has been recently argued [7] that self-reflection in RTHS can make NPCs pathfind in a more natural, human-like fashion. It will be of interest to run a user study to see if adding the anxious-learning mechanism we presented here indeed makes the NPCs more visibly human. Can the anxiety be used with tactical pathfinding (e.g., via influence maps)?

VII. Conclusions

This paper suggested a new learning mechanism for real-time heuristic search agent. It allows the agent to escape heuristic depressions faster without permanently removing states from the search graph. Empirical evaluation on video-game pathfinding benchmarks demonstrates benefits for the average solution quality when the new mechanism is used by itself or in combination with expendable state marking.

Acknowledgments

We appreciate NSERC funding and help from C. MacDonald.

References

[1] R. Korf, "Real-time heuristic search," *AIJ*, vol. 42, no. 2–3, pp. 189–211, 1990.
[2] S. Koenig, "Agent-centered search," *AI Mag*, vol. 22, no. 4, pp. 109–132, 2001.
[3] N. R. Sturtevant and V. Bulitko, "Reaching the goal in real-time heuristic search: Scrubbing behavior is unavoidable," in *SoCS*, 2014, pp. 166–174.
[4] N. Sturtevant and V. Bulitko, "Scrubbing during learning in real-time heuristic search," *JAIR*, vol. 57, pp. 307–343, 2016.
[5] G. Sharon, N. R. Sturtevant, and A. Felner, "Online detection of dead states in real-time agent-centered search," in *SoCS*, 2013, pp. 167–174.
[6] C. Hernandez, A. Botea, J. A. Baier, and V. Bulitko, "Online bridged pruning for real-time search with arbitrary lookaheads," in *IJCAI*, 2017, pp. 510–516.
[7] V. Bulitko, "Effects of self-knowledge: Once bitten twice shy," in *EXAG/AIIDE*, 2017, pp. 26–33.
[8] R. S. Lazarus and J. R. Averill, "Emotion and cognition: With special reference to anxiety," in *Anxiety: Current Trends in Theory and Research*, 1972, vol. 2, pp. 241–290.
[9] V. Bulitko, "Per-map algorithm selection in real-time heuristic search," in *AIIDE*, 2016, pp. 143–148.
[10] V. Bulitko, S. Solomon, J. Gratch, and M. van Lent, "Modeling culturally and emotionally affected behavior," in *AIIDE*, 2008, pp. 10–15.
[11] S. Campano, N. Sabouret, E. de Sevin, and V. Corruble, "An evaluation of the COR-E computational model for affective behaviors," in *AAMAS*, 2013, pp. 745–752.
[12] J. Feldmaier and K. Diepold, "Path-finding using reinforcement learning and affective states," in *RO-MAN*, 2014, pp. 543–548.
[13] V. Bulitko and M. Brown, "Flow maximization as a guide to optimizing performance: A computational model," *ACS*, vol. 2, pp. 239–256, 2012.
[14] J. Schmidhuber, "Curious model-building control systems," *IJCNN*, pp. 1458–1463, 1991.
[15] N. Rivera, J. A. Baier, and C. Hernández, "Incorporating weights into real-time heuristic search," *AIJ*, vol. 225, pp. 1–23, 2015.
[16] V. Bulitko, "Searching for real-time search algorithms," in *SoCS*, 2016, pp. 121–122.
[17] N. R. Sturtevant, "Benchmarks for grid-based pathfinding," *T-CIAIG*, vol. 4, no. 2, pp. 144 – 148, 2012.
[18] C. Hernández and J. A. Baier, "Avoiding and escaping depressions in real-time heuristic search," *JAIR*, vol. 43, pp. 523–570, 2012.
[19] A. Loreggia, Y. Malitsky, H. Samulowitz, and V. A. Saraswat, "Deep learning for algorithm portfolios," in *AAAI*, 2016, pp. 1280–1286.
[20] D. Sigurdson and V. Bulitko, "Deep learning for real-time heuristic search algorithm selection," in *AIIDE*, 2017, pp. 108–114.
[21] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *NIPS*, 2012, vol. 25, pp. 1097–1105.
[22] S. Koenig and X. Sun, "Comparing real-time and incremental heuristic search for real-time situated agents," *AAMAS*, vol. 18, no. 3, pp. 313–341, 2009.
[23] V. Bulitko and A. Sampley, "Weighted lateral learning in real-time heuristic search," in *SoCS*, 2016, pp. 10–18.

Analysis of Self-Adaptive Monte Carlo Tree Search in General Video Game Playing

Chiara F. Sironi and Mark H. M. Winands
Games & AI Group, Department of Data Science and Knowledge Engineering
Maastricht University, Maastricht, The Netherlands
{c.sironi, m.winands}@maastrichtuniversity.nl

***Abstract*—A purpose of General Video Game Playing (GVGP) is to create agents capable of playing many different real-time video games. Instead of using a fixed general strategy, a challenging aspect is devising strategies that adapt the search to each video game being played. Recent work showed that on-line parameter tuning can be used to adapt Monte-Carlo Tree Search (MCTS) in real-time. This paper extends prior work on Self-adaptive Monte-Carlo Tree Search (SA-MCTS) by further testing one of the previously proposed on-line parameter tuning strategies, based on the N-Tuple Bandit Evolutionary Algorithm (NTBEA). Results show that, both for a simple and a more advanced MCTS agent, on-line parameter tuning has impact on performance only for a few GVGP games. Moreover, an informed strategy as NTBEA shows a significant performance increase only in one case. In a real-time domain as GVGP, advanced parameter tuning does not seem very promising. Randomizing pre-selected parameters for each simulation appears to be a robust approach.**

***Index Terms*—Monte-Carlo tree search, self-adaptive search, general video game playing, on-line parameter tuning**

I. Introduction

In GVGP [1] there is the need to devise general search approaches able to deal in real-time with many heterogeneous video games. Moreover, such approaches cannot exploit game-specific and prior knowledge. MCTS with its variations and enhancements [2] is one of the most commonly used techniques in GVGP. MCTS is often controlled by many parameters and the best parameter settings vary across games. A problem when using MCTS in GVGP is that parameters cannot be tuned in advance for the game at hand and have to be set to values that are generally good (i.e. perform overall well on a reference set of games, possibly heterogeneous).

Recently, methods to tune search parameters on-line have been investigated. Sironi and Winands [3] propose on-line parameter tuning for General Game Playing and show that on-line tuned agents almost reach the same performance of off-line tuned agents. Sironi *et al.* [4] apply on-line parameter tuning to obtain a Self-adaptive MCTS (SA-MCTS) strategy for GVGP and show that, when SA-MCTS is implemented in the SampleMCTS agent of the General Video Game AI framework (GVGAI) the win rate in a few games is increased.

This paper extends prior work on SA-MCTS. It considers one of the allocation strategies that performed best in [4], the one based on the N-Tuple Bandit Evolutionary Algorithm (NTBEA). This is compared with a less informed allocation strategy based on a Multi-Armed Bandit (MAB) and with a random allocation strategy. These allocation strategies are tested both on a simple and a more advanced MCTS agent. The effect of increasing the search budget is also investigated.

The paper is structured as follows. Section II describes the SA-MCTS approach and the tested allocation strategies. Section III discusses the results obtained by the experiments. Conclusion and future work are presented in Section IV.

II. Self-adaptive Monte-Carlo Tree Search

This section describes how parameters can be tuned on-line to obtain a self-adaptive behavior of the search. Subsection II-A describes how on-line parameter tuning is integrated with MCTS, while Subsection II-B describes three strategies that decide how to allocate the available samples to evaluate different parameter value combinations.

A. Integration of On-line Parameter Tuning with MCTS

Figure 1 gives an overview of SA-MCTS. The central box shows the four phases of standard MCTS: *selection*, *expansion*, *play-out* and *backpropagation*. To tune parameters on-line two more phases are added to the search: (i) an initial phase where an allocation strategy chooses which combination of parameter values will control the next simulation, and (ii) a final phase where the reward obtained by the simulation is used to update statistics about the chosen combination of parameters.

B. Allocation Strategies

An allocation strategy decides how to allocate the available samples to the feasible parameter values that need to be evaluated. This section describes the three considered allocation strategies: *Random*, *Multi-Armed Bandit* (MAB) [5], *and N-Tuple Bandit Evolutionary Algorithm* (NTBEA) [6].

1) Random: before each MCTS simulation this allocation strategy selects a parameter combination uniformly at random from a set of manually pre-selected feasible values. This means that it does not need to collect any statistics about the performance of the combinations. This study includes a random strategy to verify how the other more informed allocation strategies would compare to one that uses no information collected on-line to select parameter values. Note that by pre-selecting the set of values we are still giving information to the strategy, and its performance will depend on how good these values are.

978-1-5386-4359-4/18/$31.00 ©2018 IEEE

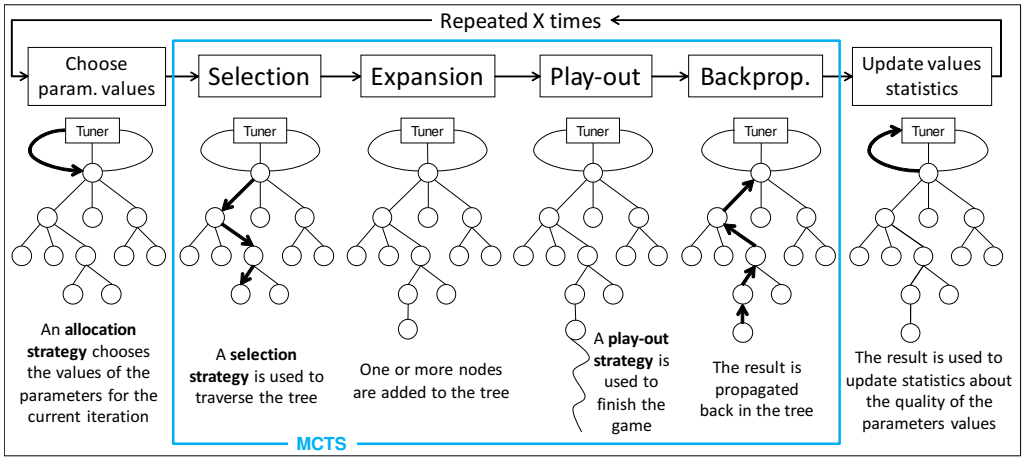


Fig. 1. Interleaving on-line tuning with MCTS

2) MAB: this allocation strategy considers the problem as a Multi-Armed Bandit [5], where each arm corresponds to one of the combinations of parameter values. Before each simulation a combination of parameter values $\vec{p}^* = \langle p_1, ..., p_d \rangle$ is selected with UCB1 as follows:

$$\vec{p}^* = \underset{\vec{p} \in \mathcal{P}}{\operatorname{argmax}} \left\{ \overline{Q}(\vec{p}) + C_{\text{MAB}} \times \sqrt{\frac{\ln N}{N_{\vec{p}}}} \right\} .$$

Here, $\mathcal{P}$ is the set of all combinations of parameters, $\overline{Q}(\vec{p})$ is the average reward obtained by all simulations controlled by combination $\vec{p}$ (normalized in $[0, 1]$), C_{MAB} is the *exploration constant*, N is the total number of simulations performed so far, and $N_{\vec{p}}$ is the number of simulations performed so far that were controlled by the parameter combination $\vec{p}$. When selecting the combination that maximizes the UCB1 value, unexplored combinations are assigned a predefined value fpu (i.e. *first play urgency*). For the experiments presented in Section III C_{MAB} is set to 0.7 and fpu is set to 1.0.

A limitation of this strategy is that it ignores the combinatorial structure of the search space. When choosing parameter values, it does not consider that good values in a combination might be good in general or in other combinations.

3) NTBEA: this allocation strategy is based on the algorithm proposed by Lucas *et al.* [6]. Two main components of NTBEA can be distinguished: an *evolutionary algorithm*, and an *N-Tuple fitness landscape model* (LModel). The evolutionary algorithm considers each combination of parameters as an individual and each single parameter as a gene. It starts with a randomly generated parameter combination and evolves it over time using statistics collected in LModel (e.g. average reward and number of visits of tuples of parameters) to decide which combination should be evaluated next. More precisely, the evolutionary algorithm repeats the following steps:

1) Use the current combination $\vec{p}$ to control an MCTS simulation.
2) Use the reward obtained by the MCTS simulation to update the statistics for parameter tuples in *LModel*.
3) Generate x neighbors of $\vec{p}$, each by mutating the value of a randomly selected parameter in $\vec{p}$.
4) Evaluate each of the x neighbors using *LModel* to compute an estimate of their UCB1 value.
5) Set the neighbor with the highest estimated UCB1 value as the current combination.

More details on how to use LModel to compute the estimate of the UCB1 values can be found in [6]. For the experiments presented in Section III x is set to 5 and the exploration constant C_{NTBEA} used to compute UCB1 values is set to 0.7.

III. Experiments

This section presents an analysis of SA-MCTS under different conditions. Subsection III-A describes the experimental setup and Subsection III-B reports and analyses the results.

A. Setup

SA-MCTS is evaluated in the single-player track of the GVGAI framework [1]. The allocation strategies tune feasible preselected parameters on-line for the following MCTS agents:

- SampleMCTS: the MCTS agent provided in the framework. This agent implements a simple version of MCTS that uses UCB1 as selection strategy and a random play-out strategy. For this agent we tune the UCB1 exploration constant C with values in $\{0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2.0\}$, and the maximum search depth D with values in $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15\}$. When this agent is not tuned on-line it uses the following fixed parameter setting: $C = 1.4$, $D = 10$.
- MaastCTS2: the winner of the 2016 GVGAI Single-Player Planing Championship [7]. It implements MCTS with UCB1 selection strategy and a series of enhancements, among which the use of *Progressive History* in the selection strategy and the *N-Gram Selection Technique* (NST) as play-out strategy. For this agent we tune the UCB1 exploration constant C with values in $\{0.2, 0.4, 0.5, 0.6, 0.7, 0.8, 1.0, 1.2, 1.4, 1.6, 1.8, 2.0\}$,

the *Progressive History* weight W with values in $\{0.1, 0.25, 0.5, 1, 3, 5, 7.5, 10, 20, 50\}$, the NST probability of playing a random action ϵ with values in $\{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$, the minimum number of visits for an N-gram to be considered in the computation K with values in $\{1, 3, 5, 7, 10, 15, 20, 30, 50\}$, and the maximum used N-gram length N with values in $\{1, 2, 3, 4, 5\}$. When this agent is not tuned on-line it uses the following fixed parameter setting: $C = 0.6$, $W = 1$, $\epsilon = 0.5$, $K = 7$, $N = 3$.

All agents do not keep any knowledge between game runs, so both the game tree built by MCTS and the parameter statistics collected by the allocation strategies are reset.

Four series of experiments were performed. The first series of experiments tests the SAMPLEMCTS agent with fixed parameter values, and three SAMPLEMCTS agents with values tuned on-line by each of the allocation strategies. The game tick is set to $40ms$ as in the GVGAI competition, and the agents are tested on 20 single-player games. The second series of experiments tests the MAASTCTS2 agent with fixed parameter values, and three MAASTCTS2 agents with values tuned on-line by each of the allocation strategies. The game tick is set to $40ms$ and the agents are tested on the same 20 single-player games. The third series of experiments is the same as the second, but the game tick is set to $100ms$ and only 10 of the initial 20 games are used. Games for which MAASTCTS2 is performing close to 100% are excluded. Only games for which there is more room for improvement when given more search time are kept. The fourth series of experiments tests the performance of SAMPLEMCTS and MAASTCTS2 when their parameters are set to fixed values expected to be sub-optimal for the games. These values are $C = 0$, $D = 1$ for SAMPLEMCTS, and $C = 0$, $W = 0$, $\epsilon = 0$, $K = \infty$, $N = 1$ for MAASTCTS2.

Results presented below always report the win percentage of the agent with 95% confidence interval. The win percentage is computed by playing all 5 levels of each game for 100 times, obtaining a total of 500 samples per game per agent.

B. Results

Tables I, II, III and IV show results obtained with the first, second, third and fourth series of experiments, respectively.

What is more interesting to observe for the first two series of experiments is that the random strategy seems to achieve in most of the games the same performance as the agent with fixed parameters. In addition, for almost all the games neither the MAB nor the NTBEA allocation strategy seem to perform better than the random strategy.

A combination of three factors might explain these results. Firstly, the manually constructed sets of parameter values might be mostly reasonable for all the games and might not contain particularly bad values. This means that for games for which the fixed parameter settings are sub-optimal, randomization will instead be able to control most of the search with optimal values. This is probably what happens in Table I for *Chase* and *Crossfire*, for which SAMPLEMCTS with the random allocation strategy seems to have a better performance than SAMPLEMCTS with fixed parameters. Secondly, the number of simulations that the agents can perform might be too small (usually a few dozen of simulations per tick, with a $40ms$ tick duration). A small number of simulation might have two implications: (i) the allocation strategies cannot find optimal values early enough in the game to make a difference in the performance, and (ii) even if there are sub-optimal values among the feasible ones, the number of simulations controlled by them is not high enough to be detrimental. Thirdly, the continuous change of parameter values selected by the allocation strategies, especially by the random one, might cause more diversity in the simulations. A more diversified search might actually be beneficial to tackle GVGP games.

From the third series of experiments (Table III) we can see that a longer search time significantly increases the performance of MAASTCTS2 in many of the games. With more search time there are a few games (*Bait*, *Chase* and *Survive Zombies*) for which the performance of the agent tuned with the random strategy seems to decrease. Also the overall performance of the agent tuned with MAB seems inferior to the other agents, as we would expect. The MAB allocation strategy

TABLE I
WIN PERCENTAGE OF SAMPLEMCTS WITH FIXED PARAMETERS AND SAMPLEMCTS WITH PARAMETERS TUNED ON-LINE BY EACH OF THE ALLOCATION STRATEGIES. GAME TICK IS SET TO *40ms*

Game	SAMPLEMCTS			
	Fixed parameters	Random	MAB	NTBEA
Aliens	**100.0**(±0.00)	**100.0**(±0.00)	99.6(±0.55)	**100.0**(±0.00)
Bait	6.6(±2.18)	**8.0**(±2.38)	7.6(±2.33)	6.8(±2.21)
Butterflies	95.2(±1.88)	94.4(±2.02)	95.2(±1.88)	**95.6**(±1.80)
CamelRace	4.2(±1.76)	5.6(±2.02)	**5.8**(±2.05)	3.8(±1.68)
Chase	3.2(±1.54)	**6.6**(±2.18)	**6.6**(±2.18)	6.2(±2.12)
Chopper	**91.4**(±2.46)	86.2(±3.03)	86.0(±3.04)	89.2(±2.72)
Crossfire	4.2(±1.76)	8.8(±2.49)	8.6(±2.46)	**14.2**(±3.06)
DigDug	0.0(±0.00)	0.0(±0.00)	0.0(±0.00)	0.0(±0.00)
Escape	0.2(±0.39)	0.8(±0.78)	**4.2**(±1.76)	0.2(±0.39)
HungryBirds	5.4(±1.98)	4.6(±1.84)	**5.6**(±2.02)	4.4(±1.80)
Infection	97.0(±1.50)	97.6(±1.34)	**98.0**(±1.23)	97.8(±1.29)
Intersection	**100.0**(±0.00)	**100.0**(±0.00)	**100.0**(±0.00)	**100.0**(±0.00)
Lemmings	0.0(±0.00)	0.0(±0.00)	0.0(±0.00)	0.0(±0.00)
MissileCommand	60.4(±4.29)	**63.0**(±4.24)	61.6(±4.27)	60.0(±4.30)
Modality	27.0(±3.90)	27.0(±3.90)	27.4(±3.91)	**28.2**(±3.95)
PlaqueAttack	91.8(±2.41)	**93.4**(±2.18)	87.6(±2.89)	92.8(±2.27)
Roguelike	0.0(±0.00)	0.0(±0.00)	**0.2**(±0.39)	0.0(±0.00)
SeaQuest	**55.0**(±4.37)	50.4(±4.39)	49.6(±4.39)	51.6(±4.38)
SurviveZombies	41.0(±4.32)	**42.6**(±4.34)	37.6(±4.25)	42.2(±4.33)
WaitForBreakfast	15.4(±3.17)	**17.6**(±3.34)	15.8(±3.20)	13.2(±2.97)
Avg Win%	39.9(±0.96)	**40.3**(±0.96)	39.9(±0.96)	**40.3**(±0.96)

TABLE II
WIN PERCENTAGE OF MAASTCTS2 WITH FIXED PARAMETERS AND MAASTCTS2 WITH PARAMETERS TUNED ON-LINE BY EACH OF THE ALLOCATION STRATEGIES. GAME TICK IS SET TO *40ms*

Game	MAASTCTS2			
	Fixed parameters	Random	MAB	NTBEA
Aliens	**100.0**(±0.00)	**100.0**(±0.00)	99.6(±0.55)	**100.0**(±0.00)
Bait	**31.8**(±4.09)	30.4(±4.04)	22.0(±3.63)	31.6(±4.08)
Butterflies	98.6(±1.03)	99.2(±0.78)	99.4(±0.68)	**100.0**(±0.00)
CamelRace	**44.4**(±4.36)	41.0(±4.32)	39.2(±4.28)	42.4(±4.34)
Chase	28.0(±3.94)	**30.4**(±4.04)	20.2(±3.52)	26.8(±3.89)
Chopper	**99.8**(±0.39)	**99.8**(±0.39)	99.6(±0.55)	99.6(±0.55)
Crossfire	**31.8**(±4.09)	27.4(±3.91)	17.2(±3.31)	28.4(±3.96)
DigDug	1.6(±1.10)	1.2(±0.96)	**1.8**(±1.17)	1.2(±0.96)
Escape	93.4(±2.18)	**94.6**(±1.98)	80.4(±3.48)	92.2(±2.35)
HungryBirds	**100.0**(±0.00)	**100.0**(±0.00)	**100.0**(±0.00)	**100.0**(±0.00)
Infection	**100.0**(±0.00)	99.8(±0.39)	99.8(±0.39)	**100.0**(±0.00)
Intersection	**100.0**(±0.00)	**100.0**(±0.00)	**100.0**(±0.00)	**100.0**(±0.00)
Lemmings	0.0(±0.00)	0.0(±0.00)	0.0(±0.00)	0.0(±0.00)
MissileCommand	**96.8**(±1.54)	96.6(±1.59)	92.6(±2.30)	94.6(±1.98)
Modality	25.6(±3.83)	24.6(±3.78)	32.0(±4.09)	**41.0**(±4.32)
PlaqueAttack	94.8(±1.95)	95.0(±1.91)	94.2(±2.05)	**95.8**(±1.76)
Roguelike	4.6(±1.84)	**4.8**(±1.88)	0.8(±0.78)	3.2(±1.54)
SeaQuest	**58.4**(±4.32)	53.6(±4.38)	53.8(±4.37)	54.6(±4.37)
SurviveZombies	42.4(±4.34)	**42.8**(±4.34)	38.2(±4.26)	40.8(±4.31)
WaitForBreakfast	**99.0**(±0.87)	98.4(±1.10)	98.0(±1.23)	98.0(±1.23)
Avg Win%	**62.6**(±0.95)	62.0(±0.95)	59.4(±0.96)	62.5(±0.95)

TABLE III
WIN PERCENTAGE OF MAASTCTS2 WITH FIXED PARAMETERS AND MAASTCTS2 WITH PARAMETERS TUNED ON-LINE BY EACH OF THE ALLOCATION STRATEGIES. GAME TICK IS SET TO *100ms*

Game	MAASTCTS2			
	Fixed parameters	Random	MAB	NTBEA
Bait	**51.8**(±4.38)	40.4(±4.31)	31.2(±4.07)	36.6(±4.23)
CamelRace	**95.8**(±1.76)	92.2(±2.35)	85.4(±3.10)	90.8(±2.54)
Chase	**56.2**(±4.35)	50.4(±4.39)	43.0(±4.34)	51.6(±4.38)
Crossfire	**84.8**(±3.15)	83.2(±3.28)	68.2(±4.09)	81.8(±3.39)
DigDug	0.0(±0.00)	0.2(±0.39)	**0.4**(±0.55)	0.0(±0.00)
Lemmings	0.0(±0.00)	0.0(±0.00)	0.0(±0.00)	0.0(±0.00)
Modality	26.2(±3.86)	25.6(±3.83)	38.8(±4.28)	**40.4**(±4.31)
Roguelike	**32.6**(±4.11)	30.6(±4.04)	28.4(±3.96)	32.0(±4.09)
SeaQuest	58.6(±4.32)	56.0(±4.36)	**61.8**(±4.26)	56.2(±4.35)
SurviveZombies	**49.0**(±4.39)	46.0(±4.37)	44.8(±4.36)	45.4(±4.37)
Avg Win%	**45.5**(±1.38)	42.5(±1.37)	40.2(±1.36)	43.5(±1.37)

TABLE IV
WIN PERCENTAGE OF SAMPLEMCTS AND MAASTCTS2 WITH SUB-OPTIMAL FIXED PARAMETER VALUES. GAME TICK IS SET TO *40ms*

Game	SAMPLEMCTS$_{\mathrm{SUB}}$	MAASTCTS2$_{\mathrm{SUB}}$
Aliens	67.0(±4.13)	100.0(±0.00)
Bait	6.0(±2.08)	23.2(±3.70)
Butterflies	66.8(±4.13)	98.8(±0.96)
CamelRace	3.0(±1.50)	32.0(±4.09)
Chase	3.6(±1.63)	29.0(±3.98)
Chopper	0.0(±0.00)	98.4(±1.10)
Crossfire	10.2(±2.66)	20.2(±3.52)
DigDug	0.0(±0.00)	1.0(±0.87)
Escape	29.6(±4.01)	92.4(±2.33)
HungryBirds	1.8(±1.17)	99.4(±0.68)
Infection	95.0(±1.91)	100.0(±0.00)
Intersection	100.0(±0.00)	100.0(±0.00)
Lemmings	0.0(±0.00)	0.0(±0.00)
MissileCommand	32.4(±4.11)	94.2(±2.05)
Modality	17.0(±3.30)	47.0(±4.38)
PlaqueAttack	24.6(±3.78)	88.0(±2.85)
Roguelike	0.0(±0.00)	1.6(±1.10)
SeaQuest	25.0(±3.80)	58.8(±4.32)
SurviveZombies	27.4(±3.91)	36.6(±4.23)
WaitForBreakfast	58.2(±4.33)	84.8(±3.15)
Avg Win%	28.4(±0.88)	60.3(±0.96)

suffers from the high overhead of computing the UCB1 value for all parameter combinations before each simulation, and in addition does not exploit information about the combinatorial structure of the parameter space. However, even if we increase the search time to $100ms$, the results are still in line with what is observed for $40ms$. The random allocation strategy is still quite robust, and MAB and NTBEA do not seem much better.

Over all the first three series of experiments, it is still interesting to notice that there are a few games for which on-line parameter tuning seems beneficial. For the SAMPLEMCTS agent, on-line tuning with NTBEA significantly increases the performance in *Crossfire*, and on-line tuning with MAB significantly increases the performance in *Escape* (Table I). Moreover, for the MAASTCTS2 agent the performance in *Modality* is significantly increased by NTBEA when using $40ms$ per tick (Table II), and by both NTBEA and MAB when using $100ms$ per tick III. This suggest that the allocation strategies that make informed decisions have the potential to be useful even when decisions have to be made fast.

The fourth series of experiments (Table IV) seems to confirm that on-line parameter adaptation (whether randomly or with an informed strategy) might still have some benefits. We can see that the sub-optimal values for these experiments cause a decrease in performance for many of the games if compared with the fixed values used in previous experiments. Nevertheless, there are games for which the performance is higher (if not the highest over all experiments), like *Crossfire*, *Escape* and *Wait For Breakfast* for SAMPLEMCTS, and *Modality* for MAASTCTS2. This suggests that for those games the fixed values used in the first three series of experiments were non-optimal. Moreover, some good values for those games might have been left out of the pre-selected sets of values.

IV. CONCLUSION AND FUTURE WORK

This paper extended the analysis of SA-MCTS in GVGP by testing on-line parameter tuning both on a simple and a more advanced MCTS agent. The effect of increasing the search time has also been evaluated. In addition, the performance of NTBEA, one of the allocation strategies that performed best in previous work on on-line parameter tuning, has been compared with a less informed allocation strategy (MAB) and a completely random allocation strategy.

Given the obtained results, we may conclude that the use of on-line parameter tuning might be more suitable for domains where a higher number of simulations can be reached, or for domains that are more sensitive to changes in the search parameter values. Moreover, we may conclude that in a real-time context like GVGP, randomization of parameter values usually gives a robust setting for a small number of parameters, especially if we pre-select small sets of feasible values.

For future work it would be interesting to see if increasing time constraints to achieve a few thousands simulations per tick would make a difference. It would also be interesting to investigate other ways of on-line self-adaptation of the search that do not necessarily involve changing search parameters.

ACKNOWLEDGMENT

This work is funded by the Netherlands Organisation for Scientific Research (NWO) in the framework of the project GoGeneral, grant number 612.001.121.

REFERENCES

[1] D. Perez-Liebana, S. Samothrakis, J. Togelius, T. Schaul, S. M. Lucas, A. Couëtoux, J. Lee, C.-U. Lim, and T. Thompson, "The 2014 general video game playing competition," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 8, no. 3, pp. 229–243, 2016.

[2] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of Monte Carlo tree search methods," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 4, no. 1, pp. 1–43, 2012.

[3] C. F. Sironi and M. H. M. Winands, "On-line parameter tuning for Monte-Carlo tree search in general game playing," in *Workshop on Computer Games*. Springer, 2017, pp. 75–95.

[4] C. F. Sironi, J. Liu, D. Perez-Liebana, R. D. Gaina, I. Bravi, S. M. Lucas, and M. H. M. Winands, "Self-adaptive MCTS for general video game playing," in *Applications of Evolutionary Computation. EvoApplications 2018*, ser. Lecture Notes in Computer Science, K. Sim and P. Kaufmann, Eds., vol. 10784. Springer, 2018, pp. 358–375.

[5] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine Learning*, vol. 47, no. 2-3, pp. 235–256, 2002.

[6] S. M. Lucas, J. Liu, and D. Perez-Liebana, "The n-tuple bandit evolutionary algorithm for game agent optimisation," *arXiv preprint arXiv:1802.05991*, 2018.

[7] D. J. N. J. Soemers, C. F. Sironi, T. Schuster, and M. H. M. Winands, "Enhancements for real-time Monte-Carlo tree search in general video game playing," in *Computational Intelligence and Games (CIG), 2016 IEEE Conference on*. IEEE, 2016, pp. 1–8.

Game AI Research with Fast Planet Wars Variants

Simon M. Lucas
Game AI Research Group
School of Electronic Engineering and Computer Science
Queen Mary University of London

***Abstract*—This paper describes a new implementation of Planet Wars, designed from the outset for Game AI research. The skill-depth of the game makes it a challenge for game-playing agents, and the speed of more than 1 million game ticks per second enables rapid experimentation and prototyping. The parameterised nature of the game together with an interchangeable actuator model make it well suited to automated game tuning. The game is designed to be fun to play for humans, and is directly playable by General Video Game AI agents.**

I. Introduction

This short paper describes a new platform for Game AI research based on variations of the Planet Wars game. The platform has been designed from the ground up for speed and flexibility. The core game has a fast forward model and an efficiently copyable game state,[1] and is therefore ideal for statistical forward planning algorithms. Furthermore, it has many options which are all bundled into a single parameter object, so that the details of the game can be changed dynamically within different copies of the game.

This enables the effects of inaccurate forward models to be systematically investigated, and also makes it well suited to research into automated game tuning. The games are directly playable by General Video Game AI (GVGAI) agents, therefore adding games with strategic depth to that platform. This is particularly useful for extending the type of games offered by the GVGAI 2-player track [1], but can also be used for the single player tracks [2] by providing one or more fixed opponents. Finally, the game has also been designed to be fun for human players.

The last decade has seen increasing interest in testing AI on Real-Time Strategy games, with StarCraft being an obvious example. There is also an important place for simpler and faster games, which not only offer more convenient experimentation but can be easily varied to provide more general and varied AI challenges than can be offered by a single game. Recent examples include microRTS [3] and ELF [4]. Compared to microRTS and ELF, the game described in this paper runs around ten times faster due to the simpler rules of the game, and specifically due to design decisions which limit the number of game entities in play at any one time. Compared to the stripped down version described in [5] the platform in this paper offers more sophisticated game play as described below, and introduces the additional features of spinning turrets and a gravity field.

[1]Some game implementations have state variables spread throughout the code making it awkward and possibly inefficient to copy the state.

II. Planet Wars

Planet Wars is a popular casual two-player real-time strategy game with versions going under many names on various platforms. The game is a simple real-time strategy game that is fun for humans to play and provides an interesting challenge for AI. It was used for the 2010 Google AI challenge (http://planetwars.aichallenge.org/) run by the University of Waterloo in Canada [6] with great success. The game was also successfully used by Buro et al. for a Dagstuhl AI Hackathon, who also describe the rules of the standard game [7].

A. Rules

The aim of the game is for a player to take over all enemy planets by sending ships to invade them. Each planet is either owned by either player or is neutral, and each non-neutral planet spawns new ships. Good strategy involves balancing the need to take over as many planets as quickly as possible versus leaving currently owned planets with enough ships to deter invasion. The game is played out on a 2D map, and ships take time to travel between planets, hence there is an interesting spatio-temporal planning aspect to the game. Indeed, Leece and Jhala [8] specifically chose the game to study the ability of q-learning agents to deal with spatial planning.

The Map specifies how the planets are laid out in 2D space. Planets are defined by their position, their size, growth rate (proportional to their radius), and initial ownership. Hence the map alone already provides for immense variations: it is easy to generate new maps at random that will require different tactics in order to win. All aspects of the map play an important role in the decision making process. Hence, even in its default setting the game already offers significant variation, and offers a more robust test of an AI system than the individual games of the Atari 2600 platform [9], for example, even before we consider the actions of a varied set of adversaries.

Beyond this, the time taken for ships to travel between planets also affects game play, making ship speed and map size important parameters.

Further variations are possible with the user-interface. In some versions the player selects a single source and target (destination) planet for each move, in other versions multiple source planets can be selected via a drag action. However, most versions the author is aware of use the basic source / destination mechanism as a way to specify actions.

978-1-5386-4359-4/18/$31.00 ©2018 IEEE

B. Variations

All the above listed features are standard for the game. For this version we introduce a number of variations. The first two are for the purpose of efficiency. Firstly, a transporter is used to send ships between planets, so multiple ships travel on a single transporter. This is more efficient than sending them individually. Secondly, each planet is restricted to having a single transporter. This means the cost of calculating game updates via the next state function grows only linearly with the number of planets in the game. For comparison, the version used for the Google AI challenge bundled multiple ships on to a single transporter, but placed no limit on the number of transporters that could be launched at any time, other than the constraint that each one must carry at least one ship.

Beyond efficiency, two significant variations are introduced to add to the game-play: the gravity field and the rotating turret for direction selection.

The gravity field pre-computes a gravitational force that is calculated from the position and mass of each planet with mass being proportional to the planet's area (since this is 2D space). The gravity field adds significantly to the game-play: the fact that ships now follow curved trajectories means that players (especially when playing with the Slingshot actuator, see below) must carefully judge the effects of gravity when timing the release of the transporter. The curved trajectories may be more interesting to observe than linear ones, and create some dramatic tension around whether the transporter will reach the targeted planet or not.

The rotating turret adds a skill aspect for human players. Using this mechanism a player executes a long mouse press on a planet for each action. While the mouse is pressed, ships are loaded on to a transporter at each game tick. When the mouse is released, the transporter leaves heading in the current direction of the turret.

For a human player this added skill can be a source of enjoyment or of frustration, depending on the individual player and on how well the game is tuned for that player. For example, if the turret rotates too quickly then it becomes difficult to target the desired planet; if too slowly, then a long wait may be involved for the turret to point in the desired direction. For an AI agent, a slow turret can be a source of challenge since it requires planning further ahead, whereas a fast rotating turret should not make it more difficult, unless timing noise is added to the AI agent's actions.

C. Game Parameters

The game currently has 16 parameters which significantly affect the game play. At the time of writing these include the following:

- Number of planets: more planets lead to higher branching factors and more complex game-play.
- Map dimensions: the width and height of the map in pixels.
- Gravitational constant: multiplies planet mass to scale gravitational force. Higher values lead to more curved transit trajectories.
- Growth rate range: planet growth rates are sampled from a uniform distribution in this range.
- Radial Separation: as planets are placed randomly they must be at least this number of radii apart from the nearest already placed planet.
- Ship launch speed: faster launch speed means ships will tend to arrive at their destination sooner, and less influenced by the gravity field.
- Transport tax: a subtractive amount per tick that reduces the number of ships during transit (and may even take them negative, hence turn them in to opponent ships.

A screenshot of the game is shown in figure 1. This version is shown in portrait mode; the dimensions of the game can be changed as easily as any other parameter, though there are dependencies on other game parameters: for example, making the screen area smaller while retaining the same planet size will affect how many planets can be placed, and the density of the area. Maps with denser layouts may place greater importance on owning well-connected central planets. The ability to easily change screen dimensions may be true of most games with randomly generated levels, but does not hold for games that rely on file descriptions of each level, such as Pac-Man and Super Mario Bros. (for those games, level generation is a topic in its own right).

D. AI Agent Considerations

AI agents can submit at most one action per game tick, though the details depend on the actuators used, described in section III. Currently there is no fog of war: all game states are fully observable to the AI agents, though this is an obvious source of future variations. An existing variation shows players only the ownerships of each neural or opponent planet, and not the number of ships on them. Other observability variations are also possible, such as just showing a small window of the map instead of the entire map. For an example of how to explicitly vary observability in video games see the Partially Observable Pac-Man competition [10].

The branching factor of the game (average number of legal actions at each tick) depends very much on the actuator model: for the source / target model, the source planet must be owned by the player, but the destination can be any planet other than the source. The number of the planets is a parameter of the game, and during testing this has been varied between 10 and 100. Games may last for many thousands of ticks, but for experiments we often limit this to between 1,000 and 5,000 ticks, which is often enough to estimate which player is superior. Games between a strong and a weak player are often decided (and terminated) within 1,000 ticks.

We do not have statistics to support this yet, but the game seems to proceed in distinct phases. In the initial phase each player owns a small number of planets and the game appears to be finely balanced: decision on which ones to invade at this stage are important, though we do sometimes observe AI players throwing away an apparent lead. In the next phase the lead frequently fluctuates with many ships in transit and closely fought battles. This is followed by a final phase where

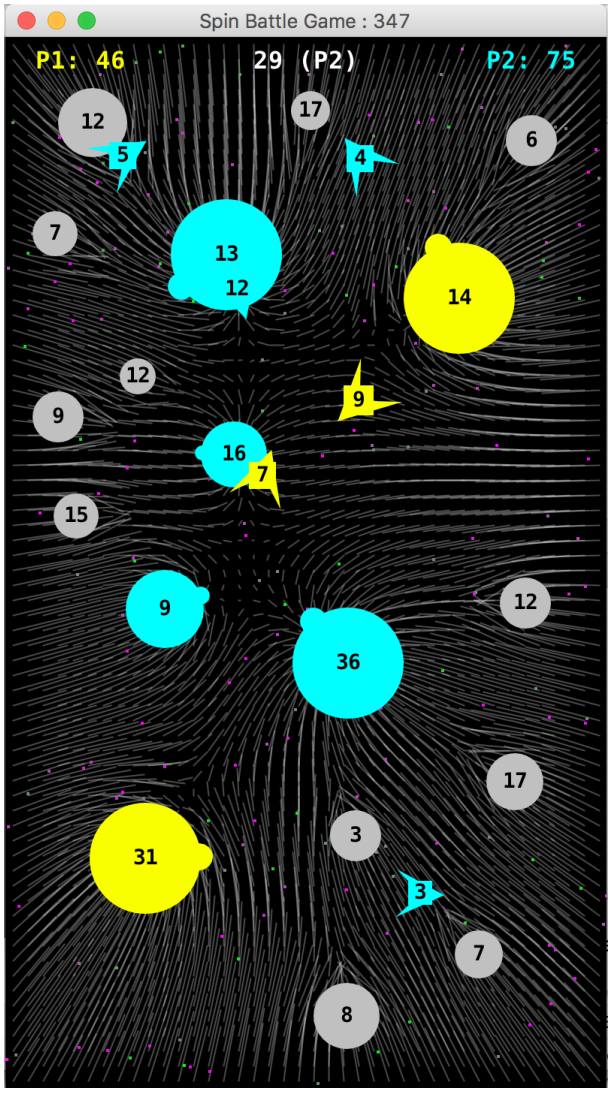


Fig. 1. Planet Wars with spinning planets and a gravity field, which both have significant effects on the game play.

one player dominates and the outcome of the game is no longer in doubt.

The game was recently tested during a graduate level AI Assisted Game Design course.[2] During the course students developed their own statistical forward planning agents and also their own tuned versions of the game, by adjusting parameters and varying the rules. All game variants were able to clearly separate the agents in to different levels of skill, measured by win rates in round robin leagues (leagues in which each player plays every other player a fixed number of times), providing evidence that the game variants have skill depth. Interestingly, the game variants interesting produced very different ranking of the AI agents.

[2]https://github.com/GAIGResearch/AIGD2

III. Implementation

The game is implemented in Java, and has been designed from the ground up to run efficiently, offer flexibility for the Game AI researcher, and also allow for easy human interaction. Enabling easy human interaction enables automatically tuned games to be tested by human players. Efficiency is important for all aspects of the research.

Key features of the design include:

- All variable parameters are stored in a single GameParameter object, and are never declared as static variables. A reference to a GameParameter object is passed to all copies of the game, but can be copied and modified on demand.
- Each planet has only a single Transporter (these are the angular space ships shown journeying through space in figure 1). This enables the cost of game state copying and updating to be kept linear rather quadratic in the number of planets. This does not seem to have a detrimental effect on the game play.
- The game state contains no circular references and can therefore be serialised into JSON for convenient storage and transmission. The largest part of the game state is the Gravity Field (which is a 2D array of Vector2D objects), but this can be nullified for serialisation and re-created on demand when needed.
- The *actuator* model has been decoupled from the game state. This means that different ways of controlling game actions can be plugged in.

Regarding the last of these points, so far two actuators have been implemented. An additional actuator based on a directional catapult is planned.

A. Source Target Actuator

Each move consists of a source planet being selected, followed by a destination planet. The move is executed only if the source planet's transporter is currently at home, and the planet is owned by the player. If these conditions are satisfied, then the ship is loaded with a percentage of the planet's ships, and launched in the direction of the target planet.

AI players currently see the number of actions at each tick as being equal to the number of planets. Low-level actions are grouped in to source-target pairs, with illegal actions (ones in which the player does not own the source planet) being ignored. A video of a rolling horizon evolution agent playing against a hand-coded heuristic agent can be viewed here: **https://www.youtube.com/watch?v=G2aoxYODs9U.**

B. Slingshot Actuator

Each move consists of a user selecting a planet for a number of game ticks. The selection only happens if the player owns the chosen planet. When the planet is deselected, the ship is launched at a standard speed in the direction the turret is facing. A video of a human player (the author) playing against a heuristic agent player can be viewed here: **https://www.youtube.com/watch?v=y2q5VW8kS8k.**

TABLE I
SPEED OF KEY OPERATIONS IN UNITS OF THOUSANDS OF OPERATIONS PER SECOND (IMAC WITH 3.4 GHZ INTEL CORE I5 CPU). NOTE THAT THE GRAVITY FIELD (GF) IS JUST COMPUTED ONCE AT THE START OF EACH GAME, AFTER THE POSITION AND SIZE OF EACH PLANET HAS BEEN FIXED.

Operation	kop/s (1)	kop/s (4)
nextState	870	1,640
copy	1,600	3,230
compute GF	1	2

TABLE II
RESULTS OF PLAYING THREE AGENTS AGAINST EACH OTHER IN A ROUND-ROBIN LEAGUE ON 10 FIXED MAPS, PLAYING EACH MAP TWICE, SO 60 GAMES IN TOTAL. SEE TEXT FOR DESCRIPTION OF EACH AGENT. THE TABLE SHOWS THE ROW AGENT WINS AGAINST THE COLUMN AGENT, WITH THE RIGHTMOST COLUMN SHOWING THE TOTAL NUMBER OF WINS FOR THE ROW AGENT.

	RHEA	MCTS	Rand	Wins
RHEA	-	18	20	38
MCTS	2	-	17	19
Rand	0	3	-	3

C. Timing Results

Table I shows the timing results for a single thread and four threads running on an iMac with core m5 processor. The software includes the facility to run multiple games in multiple threads, or different game agents in different threads, enabling speeds in excess of 1.6 million ticks per second when running four threads simultaneously. For comparison, ELF and microRTS offer speeds of around $50k$ ticks per second when running single-threaded, meaning that this game is more than 10 times faster. The games are obviously different so comparing timings may seem unfair, but the point of the comparison is to highlight the speed offered by our platform and the rapid generation of results that this enables, even on a standard laptop or desktop computer.

IV. GENERATING AI AGENT RESULTS

The software distribution includes the following experiments ready to run, including human versus AI and AI versus AI. For human versus AI, the AI controllers are generally superior to casual human players, but their intelligence can easily be varied and decreased if necessary by reducing the simulation budget or the sequence or rollout length, in order to provide an easier challenge.

For AI versus AI, the game has been tested on the graduate student course mentioned above, and also for this paper a small but representative test was run using 3 different controllers, reported in table II. RHEA is the rolling horizon agent from Lucas et al [5], but with a sequence length of 200 and 20 iterations per move. The MCTS agent is the sample agent from the GVGAI 2-player track, but with a rollout length set to 100 and iterations per move set to 40. Hence both RHEA and MCTS used an evaluation budget of approximately $4,000$ game ticks per move. Rand is a uniform random agent. Games were limited to $2,000$ moves but often terminated with a win before reaching the limit. Running the 60 games for this mini-tournament took less than 5 minutes on the iMac computer described above. A follow-up paper will investigate these results more thoroughly, but recent experiments on this and on some other games show rolling horizon evolution frequently outperforming MCTS.

V. CONCLUSIONS

The Planet Wars platform described in this paper provides a useful addition to a growing number of games designed for AI research. The platform is efficient and well suited to testing statistical forward planning algorithms such as Monte Carlo Tree Search and Rolling Horizon Evolution. The game already has sixteen parameters that can be varied in order to significantly affect the game play and provide a thorough test of the strengths and weaknesses of the competing agents.

Future work includes introducing further variations while retaining the speed of the game, integration with AI environments such as OpenAI Gym, and additional actuator models such as a directional catapult. The speed of the game and its extensive parameter set also make it well suited to automated game tuning [11], [12].

REFERENCES

[1] R. D. Gaina, A. Coutoux, D. J. N. J. Soemers, M. H. M. Winands, T. Vodopivec, F. Kirchgener, J. Liu, S. M. Lucas, and D. Perez-Liebana, "The 2016 two-player gvgai competition," *IEEE Transactions on Games*, vol. 10, no. 2, pp. 209–220, June 2018.

[2] D. Perez-Liebana, J. Liu, A. Khalifa, R. D. Gaina, J. Togelius, and S. M. Lucas, "General Video Game AI: a Multi-Track Framework for Evaluating Agents, Games and Content Generation Algorithms," *arXiv preprint arXiv:1802.10363*, 2018.

[3] S. Ontanón, "Combinatorial multi-armed bandits for real-time strategy games," *Journal of Artificial Intelligence Research*, vol. 58, pp. 665–702, 2017.

[4] Y. Tian, Q. Gong, W. Shang, Y. Wu, and L. Zitnick, "ELF: an extensive, lightweight and flexible research platform for real-time strategy games," *CoRR*, vol. abs/1707.01067, 2017. [Online]. Available: http://arxiv.org/abs/1707.01067

[5] S. M. Lucas, J. Liu, and D. Perez-Liebana, "The N-Tuple Bandit Evolutionary Algorithm for Game Agent Optimisation," *arXiv preprint arXiv:1802.05991*, 2018.

[6] A. Fernández-Ares, A. M. Mora, J. J. Merelo, P. García-Sánchez, and C. Fernandes, "Optimizing player behavior in a real-time strategy game using evolutionary algorithms," in *Evolutionary Computation (CEC), 2011 IEEE Congress on*. IEEE, 2011, pp. 2017–2024.

[7] S. M. Lucas, M. Mateas, M. Preuss, P. Spronck, and J. Togelius, "Artificial and computational intelligence in games: Integration (dagstuhl seminar 15051)," in *Dagstuhl Reports*, vol. 5, no. 1. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.

[8] M. Leece and A. Jhala, "Reinforcement Learning for Spatial Reasoning in Strategy Games," 2013.

[9] M. C. Machado, M. G. Bellemare, E. Talvitie, J. Veness, M. J. Hausknecht, and M. Bowling, "Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents," *CoRR*, vol. abs/1709.06009, 2017. [Online]. Available: http://arxiv.org/abs/1709.06009

[10] P. R. Williams, D. P. Liebana, and S. M. Lucas, "Ms Pac-Man Versus Ghost Team CIG 2016 Competition," *IEEE Conference on Computational Intelligence and Games*, 2016.

[11] E. J. Powley, S. Colton, S. Gaudl, R. Saunders, and M. J. Nelson, "Semi-automated level design via auto-playtesting for handheld casual game creation," in *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, Sept 2016.

[12] K. Kunanusont, R. D. Gaina, J. Liu, D. Perez-Liebana, and S. M. Lucas, "The n-tuple bandit evolutionary algorithm for automatic game improvement," in *2017 IEEE Congress on Evolutionary Computation (CEC)*, June 2017, pp. 2201–2208.

A Refined 3D Dataset for the Analysis of Player Actions in Exertion Games

Chrysoula Varia*, Georgios Tsatiris*, Kostas Karpouzis*
*School of Electrical and Computer Engineering
National Technical University of Athens
Athens, Greece
chrysvaria@gmail.com, gtsatiris@image.ntua.gr,
kkarpou@cs.ntua.gr

Stefanos Kollias*†
†School of Computer Science
University of Lincoln
Lincoln, UK
skollias@lincoln.ac.uk

***Abstract*—Modeling and accurately analyzing human activities plays an important role, considering the rise of modern applications in human-computer interaction and, more recently, exertion games. Especially in serious exergames aimed at tutoring (e.g. sports) or rehabilitation and physiotherapy, the need for accurate detection of the human body and its motion is uncompromising. However, modern human skeleton tracking techniques suffer from a variety of issues, such as jittering and sensitivity to original conditions. In this study we show how a simple yet effective fairing pipeline on an inherently noisy dataset can produce data capable for precise experimentation with state-of-the-art human action modeling algorithms.**

***Index Terms*—exertion games, skeleton tracking, human action analysis, player actions**

I. Introduction

Exertion games form a very active research field with applications of almost interdisciplinary fashion, such as interaction design, gaming, sports, health and rehabilitation and, naturally, entertainment [1]. Many challenges arise in the field, mainly from the necessity of capturing and analyzing human motion, and transferring it into the application. In that manner, we investigate the exergames domain, mainly from the standpoint of recognizing and categorizing human activities.

Modeling of human actions find increasing use in the field of digital games and human-computer interaction [2], due to the commercial success of low-cost depth sensors, such as the Microsoft Kinect and the Intel RealSense series. Recent applied examples of serious games utilizing human actions can be found in the literature [3] and showcase the potential of human motion analysis, especially in healthcare-related games. The fact that there are many recent and prominent attempts at modeling actions for exertion gaming applications [4] shows the growing research interest in the field.

II. Human pose acquisition techniques and issues

One of the most widely applied methods to capture human motion, using vision-based techniques, is to detect the human pose in the frames of a video sequence. The pose of a subject is usually characterized by their skeleton joints, i.e. points in the frame that correspond to parts of the human body, such as the ankles, the wrists, the elbows, the head and the base of the neck.

In the context of depth sensing, skeleton joints are extracted from depth videos, using methodologies such as the ones documented in [5] and [6]. In [5], the authors proposed a methodology that is based on an object recognition approach, designing representations for intermediate body parts and transforming the pose estimation problem into a per-pixel classification problem. 3D proposals of body joints are then generated, based on the classification results. Later, in [6], a more advanced approach is proposed, based on a voting scheme to directly infer the positions of joints.

Recent prominent techniques have focused on solving the skeleton tracking problem in regular monocular color cameras instead of depth sensors. In the work presented in [7], the authors propose a deep convolutional neural network based pipeline that detects candidate body parts in the scene, as well as evaluated relations between them. It then proceeds to solve a heavily relaxed graph matching problem and extract connected skeletons for all persons detected in the scene.

A. Issues with skeleton data from depth sensors

Studies such as the one by Cosgun *et al.* [8] have investigated the accuracy of conventional depth sensing techniques with respect to the accuracy of extracted skeletons. In this study, it is claimed that average joints errors have been observed to usually be more than 5 cm. Such fluctuations in the position of a body part can be critical in accurate sensing of the human position and motion, especially in healthcare applications. Another study [9] places that error at about 10cm, especially in general, non-constrained postures. Moreover, depth-based skeletal tracking struggles with occluding body parts and objects in the scene.

Another issue, found both in depth-based and RGB-based skeletal tracking, is the sensitivity of the result with respect to initialization, especially when dealing with unconstrained scenes [10]. In other words, the way at which we begin the attempt of detecting skeleton joints affects the final result. Repetition of an action, or background processing of the sequence may leverage this issue and obtain more refined results.

978-1-5386-4359-4/18/$31.00 ©2018 IEEE

B. The current state of the THETIS dataset

Our work in this paper focuses on the THETIS dataset [11]. It is comprised by 31 amateurs and 24 experienced players performing tennis shots. Data capturing was done using a Kinect sensor placed in front of the subjects. The action classes in the dataset are: backhand with two hands, backhand, backhand slice, backhand volley, forehand flat, forehand open stands, forehand slice, forehand volley, service flat, service kick, service slice and smash.

Although THETIS includes RGB and depth videos (scaled to grayscale), original (raw) depth information can be obtained using the unconstrained ONI files. So skeleton tracking needs to be handled from scratch. Another major issue, though, is that, in some cases, other subjects interfere in the scene, as data capturing took place mostly in a gym. Samples from the THETIS dataset can be seen in figure 1.

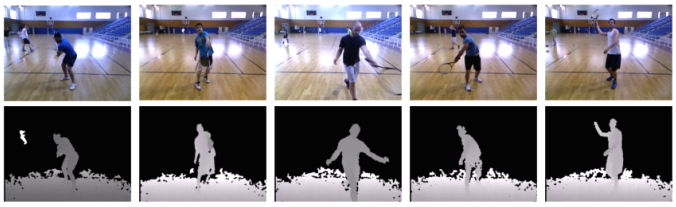


Fig. 1. RGB and depth samples from the THETIS database for the backhand, flat service, forehand flat, slice service and smash action classes.

III. Denoising the THETIS dataset

When dealing with a pre-existing dataset, assumptions as to what needs to be detected in the scene can be made, considering the specifics of the problem. In the THETIS dataset, this refers to the detection of the skeletal joints of the tennis player in the foreground of the scene. Joints are extracted in a per-frame fashion, using the method by Shotton *et al.* [6]. As it has already been discussed, there exist cases of individuals interfering during the execution of the action by the subject. However, we can expect that the tennis player will eventually be the subject detected in the vast majority of the frames of the sequence. We call this subject the *prominent subject*. Other persons detected in the sequence are subsequently rejected.

The complete pipeline is presented in algorithm 1. Due to the fact that a re-run of the sequence or another repetition of the action will, in most cases, give results of varying precision (as discussed previously), we leverage many iterations of the same sequence to obtain the final, refined annotations. In every iteration of the algorithm, we calculate the value of a factor to which the current iteration will impact the final result. This factor depicts the number of times the prominent subject has been detected in a specific frame. Starting from 0, we increment the factor in every iteration in which the prominent subject has been detected in this frame.

Ultimately, in the i-th iteration, the coordinates of joint J in a particular frame are updated using equation 1. J^i_{final} is the updated (final) values of coordinate vector J in the i-th iteration, J^{i-1}_{final} are the final values calculated by the previous iteration and J^i_{new} are the latest values in the current iteration, before updating. Essentially, what is achieved is an averaging scheme, tailored to the needs of a skeletal joint dataset. The algorithm finishes when the contribution of the latest iteration becomes negligible, i.e. $\frac{1}{\boldsymbol{factor}} < threshold$, where $threshold$ is a predefined value.

$$\boldsymbol{J^i_{final}} = \frac{\boldsymbol{factor}-1}{\boldsymbol{factor}}\boldsymbol{J^{i-1}_{final}} + \frac{1}{\boldsymbol{factor}}\boldsymbol{J^i_{new}} \quad (1)$$

Algorithm 1 Pseudocode for the kinect-based skeletal information refinement pipeline

```
initialize factor vector
threshold ← a
repeat
  repeat
    detect all subjects
    for all detected subjects do
      increment number of occurrences
    end for
  until end of video sequence
  determine prominent subject
  for all video sequence frames do
    if prominent subject present in frame then
      recalculate prominent subject's joint positions in
      frame, based on equation 1
      factor++
    end if
  end for
until 1/factor < threshold
```

An alternative to this pipeline would be another averaging filter such as median filtering. This filter is a non-linear digital filtering technique, mainly focused on smoothing out individual noise and spikes. However, as this is not particularly the case in skeleton tracking, where noise is mostly uniformly distributed, this may not have given the desired result. In figure 2, we can see a set of refined skeleton joints from a sample of the "backhand" action of THETIS. Finally, the complete set of refined skeletal annotations can be found in the THETIS website[1].

IV. Experimental results

In order to verify the robustness of the produced annotations, we used the refined skeletal data as input for two different action analysis scenarios with wide applicability in the world of human-computer interaction and exertion games. The first scenario is that of generic activity recognition and is well represented by studies such as the ones presented in [4] and [12]. The second is the qualitative assessment of actions in context. A typical study in the field is the one in [13], which focuses on characterizing the level of expertise of tennis players.

[1] http://thetis.image.ece.ntua.gr/

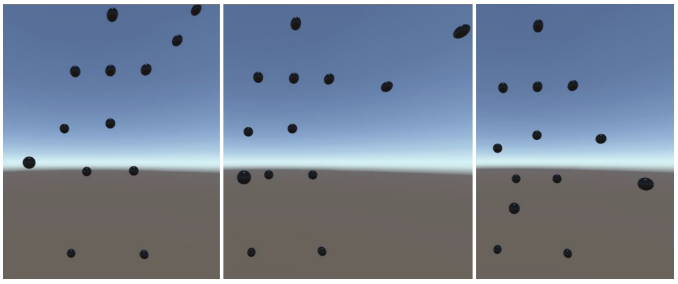


Fig. 2. Samples of extracted and refined skeleton joints of a "backhand" action from the THETIS dataset, as shown in a Unity-based application window. Viewpoint (perspective) distortions are visible.

A. Action recognition using Activity Feature Vectors

In the recent work by Cippitelli *et al.* [12], an action recognition pipeline was implemented, based on skeleton data, as the authors deem them as the most compact and representative form of the human presence in a scene. The algorithm is comprised by the following steps:

- Extraction of feature vectors representing human postures, using raw skeletal data. Essentially, a set of spatial (distance) features is calculated and normalized, based on the distance of the joints from the center of the torso.
- Selection of the most important postures for every activity class, using a clustering algorithm and picking the cluster centers as representative feature vectors.
- Computation of the final Activity Feature Vectors representing the activity as a whole. Each activity posture is characterized by the clustering of the previous step and the final Feature Vector contains the most important postures, by the order at which they appear in the activity sequence.

This method produces relatively short feature vectors, so dimensionality reduction is not necessary in most cases. Finally, a multiclass SVM is utilized to perform the classification.

In our experimental setup, we use the leave-one-person-out cross-validation protocol described thoroughly in [4]. Using this scenario, the pipeline based on the aforementioned technique and the refined skeleton dataset achieves an average accuracy of 93.65%. Table I showcases class-specific performance.

Average accuracy of the proposed pipeline is then compared with the other published techniques tested on the skeletal subset of THETIS [11] [4]. As we can see in table II, the pipeline using the novel technique in [12] on the refined skeletal data achieves higher average performance. Most notably, the method outperforms the novel approach presented in [4], which is based on a 3D Trace transform of the spatio-temporal volume of action sequences. However, this method has shown better performance in other modalities and is therefore more extensively applicable.

B. Assessing the level of expertise of tennis players

Although generic classification of human actions is a widely investigated field, qualitative assessment of actions has remained in relative obscurity. In the context of serious games focusing at player tutoring, rehabilitation and other activity-related tasks, the ability to make assumptions based on the quality of a movement is crucial. For that reason, in the work presented in [13], simple variance-based shape descriptors are utilized to classify between amateurs and experienced players in the THETIS dataset. The level of experience is self-declared, as experienced players either have been regularly practicing tennis or have attended tennis courses. Amateurs, on the other hand, have seldom or never played tennis before. For the purpose of capturing the dataset, they executed tennis actions with the help of a tutor.

TABLE I
CLASS-SPECIFIC RESULTS OF THE ALGORITHM PRESENTED IN [12], WHEN USED ON THE REFINED THETIS SKELETON DATASET

Action class	Accuracy (%)
Backhand	0,9549
Backhand with two hands	0,9775
Backhand slice	0,9275
Backhand volley	0,9388
Forehand flat	0,9308
Forehand open	0,9678
Forehand slice	0,9147
Service flat	0,9275
Service kick	0,9275
Service slice	0,9163
Smash	0,9195
Volley	0,9356

TABLE II
AVERAGE ACCURACY COMPARISON BETWEEN FOUR PUBLISHED PIPELINES AND THE ACTIVITY FEATURE VECTORS BASED PIPELINE ON THE REFINED THETIS DATASET

Method	Accuracy (%)
STIPs with HOG/HOF descriptors [14]	54.40
Dense Trajectories: Trajectory [15]	46.84
Dense Trajectories: MBH [15]	46.84
Dence Trajectories: Combination [15]	53.08
3D CTT - Selective STIP [16] based VTFs [4]	86.06
Activity Feature Vectors [12] on refined data	**93.65**

The pipeline in [13] calculates the variance of the points in a frame from the mean point (average of all points) and constructs a variance vector characterizing each action sequence. These vectors are then used as input in a simple K-NN classification scheme. However, due to the difference in length between sequences, Dynamic Time Warping [17] is utilized in two different classification scenarios:

- As a distance metric for the K-NN algorithm, as it calculates the minimum distance between two sequences
- By time-aligning the input sequences and then using Euclidean distance as a metric.

Classification was performed in an activity class specific manner. Both scenarios showed promising class-specific results, given the simplicity of the calculated features and the complexity of the task at hand.

Similar to [4], this method uses Selective Spatiotemporal Interest Points [16] as input. STIPs in general, as originally proposed by Laptev *et al.* [14] [18] represent, in the context

TABLE III
PER-CLASS ACCURACY OF THE SHAPE DESCRIPTOR-BASED PIPELINE [13], WITH SELECTIVE STIPS [16] AND OUR REFINED SKELETAL JOINTS AS INPUT

Action class	Selective STIPs		Skeletal Joints	
	Not aligned, DTW metric	Time-aligned, Euclidean distance	Not aligned, DTW metric	Time-aligned, Euclidean distance
Backhand	74.55	64.85	**79.59**	77.55
Backhand with two hands	**70.91**	66.06	61.11	62.96
Backhand slice	64.85	**69.09**	68.52	55.56
Backhand volley	**69.09**	63.64	60.42	64.58
Forehand flat	**69.09**	58.18	53.57	53.57
Forehand open stands	**68.48**	68.48	65.45	67.27
Forehand slice	66.06	61.21	**70.91**	54.55
Forehand volley	66.06	60.61	**71.70**	62.26
Service flat	63.64	63.03	**63.83**	61.70
Service kick	**72.73**	70.91	62.22	64.44
Service slice	**67.88**	66.67	61.54	59.62
Smash	**64.85**	63.03	58.49	62.26

of human motion, points on the human body that express movement. Skeleton joints can be considered a very concise but representative subset of these points and therefore can be treated the same way.

In this experimental scenario, the refined skeleton dataset is used as input for the original algorithm in [13], instead of STIPs. Again, two classification scenarios are followed, and the results can be found in table III. It can be noted that, using the refined dataset, the algorithm demonstrates similar performance with the original STIP-based implementation. In some activity classes, the joint-based pipeline performs better. The reasons why certain classes show better results than others, as well as what part the difference between STIPs and skeletal joints has to play, form legitimate investigation directions for future studies.

V. CONCLUSION

In this work, we have noted the importance of accurate human skeleton information extraction in human-computer interaction related applications, such as exertion games. We have seen the issues that plaque average skeletal joint extraction techniques and we extended a published dataset with problematic raw data, by demonstrating a simple 3D skeleton data fairing pipeline. Experiments on two current human action related scenarios show that the refined dataset, when used with state-of-the-art algorithms, can form a basis for accurate experimentation on action-related applications.

ACKNOWLEDGMENT

This work was supported by the EC-funded project ECoWeB - Emotional Competence and Well-Being (grant agreement No 754657).

REFERENCES

[1] F. Mueller, R. A. Khot, K. Gerling, and R. Mandryk, "Exertion games," *Found. Trends Hum.-Comput. Interact.*, vol. 10, no. 1, pp. 1–86.

[2] R. Cowie, E. Douglas-Cowie, K. Karpouzis, G. Caridakis, M. Wallace, and S. Kollias, "Recognition of emotional states in natural human-computer interaction," in *Multimodal User Interfaces*. Springer, Berlin, Heidelberg, pp. 119–153.

[3] F. Deboeverie, S. Roegiers, G. Allebosch, P. Veelaert, and W. Philips, "Human gesture classification by brute-force machine learning for exergaming in physiotherapy," in *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, Sept 2016, pp. 1–7.

[4] G. Goudelis, G. Tsatiris, K. Karpouzis, and S. Kollias, "3d cylindrical trace transform based feature extraction for effective human action classification," in *2017 IEEE Conference on Computational Intelligence and Games (CIG)*, Aug 2017, pp. 96–103.

[5] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake, "Real-time human pose recognition in parts from single depth images," in *Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition*, ser. CVPR '11. IEEE Computer Society, 2011, pp. 1297–1304.

[6] J. Shotton, R. Girshick, A. Fitzgibbon, T. Sharp, M. Cook, M. Finocchio, R. Moore, P. Kohli, A. Criminisi, A. Kipman, and A. Blake, "Efficient human pose estimation from single depth images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 12, pp. 2821–2840, Dec 2013.

[7] Z. Cao, T. Simon, S.-E. Wei, and Y. Sheikh, "Realtime multi-person 2d pose estimation using part affinity fields," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[8] A. Cosgun, M. Bunger, and H. Christensen, "Accuracy analysis of skeleton trackers for safety in hri," in *Proceedings of the Workshop on Safety and Comfort of Humanoid Coworker and Assistant (HUMANOIDS), Atlanta, GA, USA*, 2013, pp. 15–17.

[9] Š. Obdržálek, G. Kurillo, F. Ofli, R. Bajcsy, E. Seto, H. Jimison, and M. Pavel, "Accuracy and robustness of kinect pose estimation in the context of coaching of elderly population," in *2012 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, Aug 2012, pp. 1188–1193.

[10] T. Schubert, A. Gkogkidis, T. Ball, and W. Burgard, "Automatic initialization for skeleton tracking in optical motion capture," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, May 2015, pp. 734–739.

[11] S. Gourgari, G. Goudelis, K. Karpouzis, and S. Kollias, "Thetis: Three dimensional tennis shots a human action dataset," in *2013 IEEE Conference on Computer Vision and Pattern Recognition Workshops*, June 2013, pp. 676–681.

[12] E. Cippitelli, S. Gasparrini, E. Gambi, and S. Spinsante, "A human activity recognition system using skeleton data from rgbd sensors," *Intell. Neuroscience*, vol. 2016, pp. 21–, Mar. 2016.

[13] G. Tsatiris, K. Karpouzis, and S. Kollias, "Variance-based shape descriptors for determining the level of expertise of tennis players," in *2017 9th International Conference on Virtual Worlds and Games for Serious Applications (VS-Games)*, Sept 2017, pp. 169–172.

[14] I. Laptev, M. Marszalek, C. Schmid, and B. Rozenfeld, "Learning realistic human actions from movies," in *2008 IEEE Conference on Computer Vision and Pattern Recognition*, June 2008, pp. 1–8.

[15] H. Wang, A. Kläser, C. Schmid, and C. L. Liu, "Action recognition by dense trajectories," in *CVPR 2011*, June 2011, pp. 3169–3176.

[16] B. Chakraborty, M. B. Holte, T. B. Moeslund, and J. Gonzlez, "Selective spatio-temporal interest points," *Computer Vision and Image Understanding*, vol. 116, no. 3, pp. 396 – 410, 2012.

[17] H. Sakoe and S. Chiba, "Dynamic programming algorithm optimization for spoken word recognition," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 26, no. 1, pp. 43–49, Feb 1978.

[18] I. Laptev, "On space-time interest points," *Int. J. Comput. Vision*, vol. 64, no. 2-3, pp. 107–123, Sep. 2005.

Evolving Agents for the Hanabi 2018 CIG Competition

Rodrigo Canaan
New York University
New York, US
rodrigo.canaan@nyu.edu

Haotian Shen
New York University
New York, US
hs2685@nyu.edu

Ruben Torrado
New York University
New York, US
rrt264@nyu.edu

Julian Togelius
New York University
New York, US
julian.togelius@nyu.edu

Andy Nealen
New York University
New York, US
nealen@nyu.edu

Stefan Menzel
Honda Research Institute Europe GmbH
Offenbach, Germany
stefan.menzel@honda-ri.de

***Abstract*—Hanabi is a cooperative card game with hidden information that has won important awards in the industry and received some recent academic attention. A two-track competition of agents for the game will take place in the 2018 CIG conference. In this paper, we develop a genetic algorithm that builds rule-based agents by determining the best sequence of rules from a fixed rule set to use as strategy. In three separate experiments, we remove human assumptions regarding the ordering of rules, add new, more expressive rules to the rule set and independently evolve agents specialized at specific game sizes. As result, we achieve scores superior to previously published research for the mirror and mixed evaluation of agents.**

***Index Terms*—artificial intelligence, games, evolutionary computation**

I. Introduction

Game-playing agents have a long tradition of serving as benchmarks for AI research. However, traditionally most of the focus has been on competitive, perfect information games, such as Checkers [1], Chess [2] and Go [3]. Cooperative games with imperfect information provide an interesting research topic not only due to the added challenges posed to researchers, but also because many modern industrial and commercial applications can be characterized as examples of cooperation between humans and machines in order to achieve a mutual goal in an uncertain environment. In this paper, we address a particularly interesting cooperative game with partial information: Hanabi [4].

Hanabi is a cooperative card game designed by Antoine Bauza released in 2010 where 2 to 5 players play with their hands facing outwards, so that only the content of the other players' hand is seen. They can only communicate through a limited number of hints, which allow a player to point to all cards of a chosen value or color in another player's hand. The objective is to build one stack for each of the five colors by playing cards in ascending value order (from 1 to 5). By discarding a card or completing a stack, one hint token is replenished to the group. The group wins if all stacks are complete (corresponding to a score of 25). The group loses a life when a card is played out of order. If all lives are used, the game ends immediately. If the draw deck runs out, the game ends after one last turn for each player. In either case, the game does not count as a win, but a partial score is computed by adding the size of each stack (or equivalently, counting the number of played cards). A more rigorous explanation of the rules can be seen in [5].

Each card has a value (also called rank or number) in the range of {1-5}, and a color (or suit) out of {B, R, Y, W, G}. From now on, a card will be referred to as (CV) where C is its color and V is its value. For example (R2) denotes a red card with value 2. If only partial information is known, we represent only the color or the value. For example, (Y) is any yellow card and (5) is any card of value 5. Keep in mind that by telling a player of a color or value, all cards not pointed are known to be of some different color or value. We call this knowledge "negative information". Although the agents discussed in this work are able to reason with negative information, we do not include it in our notation for simplicity.

We will denote the number of players in the game as #players or game size. The starting number of cards in each player's hand depends on #players: 5 cards for a game size of 2 or 3, and 4 cards for a game size of 4 or 5 players.

The game was well received by the tabletop games community, winning the *Spiel des Jahres* award in 2013 [6], and has also received attention by game AI researchers for being a challenging cooperative problem with hidden information and a limited, but well-defined communication channel.

Because both the number of hint tokens and the number of copies of each card in the deck are limited, some challenging decisions that often arise in a Hanabi match are wether to play a card with only partially known information (and risk it being unplayable) or wait for more information (using up a hint token), whether to discard a card with partial information (and risk it being the last copy of its kind in the deck) and, from the other side of the table, whether to allow other players to make these risky decisions or to use a hint token to clarify the situation. Belief about how other players operate can be a key factor in making such decisions.

978-1-5386-4359-4/18/$31.00 ©2018 IEEE

In 2018, the game will feature in a two-track competition [7] at the CIG conference. In the Mixed-Track, each agent will play with unknown agents, while in the Mirror-Track, each agent will play with copies of itself and so the behavior of other players can be assumed for the most part (except possibly for stochastic behavior and hidden information). The competition page also provides the framework used in this research, which includes a sample rule-based agent, and a vast library of rules. It also features implementations of other sample agents, such as a random agent and a MCTS agent that can optionally be equipped with a perfect model of its playing partners.

In this paper, we review existing literature on Hanabi-playing agents and present our evolutionary approach to evolving rule-based Hanabi agents. We plan to submit our best agents to both tracks of the competition.

II. RELATED WORK

Optimal play of (generalized) Hanabi has been proven to be an NP-Complete problem by Baffier *et al*. [8], even if we remove all hidden information. Due to this complexity, most research in Hanabi-playing agents [5], [9]–[11] focuses on one of two methods (or a combination of both):

- Rule-based agents, which go through an ordered list of heuristics (such as play a card that is known to be playable, discard a random card, hint a playable card, etc.) and play the first one that is applicable.
- Search-based agents, which use a model of the other players' behavior to search for the action that leads to best results, such as Monte-Carlo Tree Search (MCTS) [12], or to search for previous states that are consistent with the hints received by another player.

Osawa's [5] best performing agent enhances a sequence of rules with a search of all possible previous states that are consistent with the other player's last action.

Van den Berg *et al*. [9] optimize a rule-based agent by exhaustively searching 48 possible agents obtained by selecting one of 4 possible hint heuristics, 4 discard heuristics and 3 thresholds for playability of a card. The order of application of the heuristics was pre-defined. Each strategy was evaluated by their average score in simulated mirrored play. They also implement a MCTS agent for the game, which does not perform better than their best rule-based agent.

Eger *et al*. [10] propose an intentional rule-based agent that simulates the best hint to give, assuming a model of the other player. They also validated their agent by playing with human players (achieving lower score than in mirrored play).

All agents described so far (other than Eger's human evaluation) have in common that they assume they will be playing with agents following a similar strategy. Walton-Rivers *et al*. [11] address the problem of playing with a diverse population of agents with different strategies. They use several rule-based agents, including reimplementations of Osawa and Van den Berg, along with MCTS agents which either receive a model of the other agents' behavior or assume random play. They evaluate their agents by pairing them with a fixed set of baseline agents, and their best performing agent is called Piers, which achieves a score of 11.8 with that specific test pool. Because this paper was written by the author of the Java framework being used and by the organizer of the CIG competition, it is central to our work and will sometimes be referred to as "the original paper/article".

The best existing agents for mirrored-play are those by Cox *et al*. [13]. They treat the game as a hat-guessing game. Each hand is assigned a value by a publicly-known algorithm (e.g. 1 means "first card is playable", 2 means "first card is discardable") and each possible hint is encoded as a number that is interpreted to be the sum (mod number of players) of the values of all other players' hands. This means they manage to give a clear instruction (play or discard a specific card) to every player with a single hint. Their version of hat-guessing agent can play only games with 5 players.

Bouzy [14] generalizes hat-guessing players for game sizes 2-5. He also proposes a rule-based agent called Confidence that achieves scores of 18.16 across all game sizes. He the uses both his hat agents and the Confidence agents as evaluators for and Tree Search agents, achieving even higher scores. However, their tree search agents reportedly use 10 seconds per move on average, which makes them unsuitable for the competition as moves are expected to be returned in 40ms.

Table I shows a summary of the best agents in literature, with their evaluation mode (mirror, mixed or human play) and techniques used. These numbers give us a benchmark to measure our results against. We consider results above 18.16 for mirror play and above 11.18 for mixed play (with the same test pool as in [11]) to be an improvement for our purpose, as the hat agents require a fixed convention that is unsuitable for mixed or human play, and the Tree Search agent exceeds our time budget per move.

Note that some agents discussed in this section are specialized for a specific game size, and others are able to play with a number of cards per player different than the official rules. Column **#Players** of table I denotes what games sizes the agent is capable of playing in. We consider only the reported scores with the standard number of cards per player and, for agents able of playing multiple game sizes, we average the score across all game sizes.

III. METHODOLOGY

One of the main major gaps in current research is that rule-based agents are either hand-crafted by human experts or result of exhaustive search in a narrow search space, such as Van den Bergh *et al*. [9], who specifies a sequence of rules to be applied and then searches for the best of 48 possible selections of parameters. We propose to instead use an evolutionary algorithm to determine the rules, their order and parameters with no further human assumptions.

In this section, we rigorously define what we mean by rule-based agents and rules, how mirror and mixed evaluation are performed and how the genetic algorithm works, in order to attempt to build better agents than the human-crafted ones.

TABLE I
EXISTING RESULTS

Author	Score	Evaluation	#Players	Technique
Osawa	15.85	Mirror	2	Rule-based / Search
Van den Bergh	15.4	Mirror	3	Rule-based
Eger	17.1	Mirror	2	Rule-based / Intentional
Eger	14.99	Human Play	2	Rule-based / Intentional
Cox	24.68	Mirror	5	Hat-Guessing
Bouzy - Confidence	18.16	Mirror	2-5	Rule-Based
Bouzy - Tree Search with Confidence	20.22	Mirror	2-5	Tree Search
Walton-Rivers (Piers)	11.18	Mixed	2-5	Rule-based

A. Definitions

We define a **rule-based agent** as one that scans a list of rules ordered by priority, and immediately plays the action implied by the first applicable rule. A **rule** is defined as a function that takes a game state and the current active player and returns either a legal action, if the rule is applicable, or null if the rule is not applicable. For example, a rule that tells a player a random piece of information of a playable card would fail to return a value if no other player has a playable card or if no hint tokens are available. Otherwise, it would return the action to hint the color or value of the playable card to the player who holds that card. In trying to apply a rule, only information that is available to the active player can be used.

A common pattern in all human-created agents and most of the successful evolved agents is to execute rules roughly in the following order:

- Play a "good" card
- Give a hint to a player about a "good" card
- Discard a "useless" card
- Tell a player about a "useless" card so they can discard it

The specific implementation of a rule defines the meaning of what constitutes a good or useless card, how to break ties between two or more cards, which player to give a hint to and which hint to give if multiple pieces of information (color and value) are missing. Usually, a "good card" is a card that is (or has a high probability of being) immediately playable, whereas a "useless" card is a card that is never going to be playable again (because the stack of that color is at a higher number than the card, or all prerequisites were accidentally discarded). Some rules also care whether a card is "necessary", meaning that discarding it would prevent players from ever completing a stack.

Table IX in the appendix gives a short description of each rule used. Some rules were already natively implemented in the framework. These are classified as categories 1 and 3 in the table. Category 1 is for rules very similar to the ones described in [11], focusing on the probability (from its owner's perspective) that a card is playable , or discard-able. They were already implemented as classes in the framework. Category 2 is for rules that also appear in [11], but are not natively available as classes in the framework and had to be implemented separately by using the framework' functionality for conditional rules. We refer to the original article for an in-depth explanation of those rules. Category 3 is for native rules in the framework but do not correspond to rules in [11] and attempt to implement specific human-created strategies discussed in a strategy forum [1]. More detail on those rules can be found in the framework documentation.

Rules in category 4 were implemented by us for the purpose of this work and are described in detail in section IV.

B. Mirror and mixed evaluation

We propose the use of a genetic algorithm to determine the rules as well as their number and order to play Hanabi for different numbers of players. Agents can be evolved using two kinds of evaluation as fitness function: mirror-play performance and mixed-play performance. In **mirror-play**, an agent is paired with copies of itself, and plays n games on each of the 4 game sizes. The fitness of an agent is the average score in all $4n$ games.

For **mixed-play**, we use the same test pool of seven agents as used by [11], consisting of the following seven agents: IGGI, Internal, Outer, Legal Random, Van den Bergh, Flawed, and Piers. While we do not know the exact test pool that will be used in the competition, this set of agents was used by the authors of the competition in their previous experiments, and it contains both well-performing agents (such as Piers) and intentionally poor-performing agents (such as Legal Random and Flawed). It also contains some agents that play in a totally deterministic way and others that behave stochastically. For this reason, we expect performance with this test set to be good indicator of performance in the overall mixed competition.

For an explanation of how these agents work, see [11]. For each game size $s \in \{2, 3, 4, 5\}$, the agent being evaluated is placed in a random starting position and plays with $s - 1$ copies of the same agent from the test pool. Each player plays n games for each of the seven pairings for each of the four possible game sizes, for a total of $28n$ games per generation per agent. Once again, the fitness of an agent corresponds to the average score of all games. Note that we do not know the actual agents that will be used as test pool in the mixed track of the competition, so we use performance in the test pool of the original article as an indicator of how well our agent plays with unknown agents.

To reduce the effects of randomness, we use the same random seeds to determine the starting position and starting deck configuration for all players in a population.

C. Genetic Algorithm

The main objective of our genetic algorithm is to determine a good ordering of the existing rules in the rule set. This kind

[1] https://boardgamegeek.com/thread/1309490/finesse-bluff-reverse-finesse-explained and https://www.boardgamegeek.com/article/23427635#23427635 Access:05/15/2018

TABLE II
PARAMETERS OF THE GENETIC ALGORITHM

Parameter	Value
Population size (p)	200
Chromosome size (s)	50 or 72
Mutation rate (m)	0.1
Crossover rate (c)	0.9
Elitism count (e)	20
Tournament size (t)	5
Number of generations (G)	500
Games per generation (n)	20

TABLE III
VALIDATION RESULTS

Agent	Their Score	Our Score	s.e.m
IGGI	10.96	10.98	0.06
Outer	10.2	9.70	0.05
Legal Random	4.59	4.52	0.04
Van Den Bergh	10.88	11.02	0.06
Flawed	5.02	4.46	0.04
Piers	11.18	11.28	0.06

Note: Number of games = 4*7*400 = 11200 per agent evaluated The standard error of the mean (s.e.m.) reported here corresponds to the error in our experiment, not theirs.

of problem could be categorized as a combinatorial problem. Each rule could be represented as an integer number which does not repeat within the chromosome.

The use of Genetic Algorithm to solve combinatorial problems has been studied deeply in the literature, [15], [16]. We use the operators of swap mutation (with probability m) and ordered crossover (with probability c) , which maintain the constraints that each rule is selected only once per chromosome. Individuals are chosen for crossover using tournament selection [17] with tournament size t. An elitism count of e is enforced.

Each of the p chromosomes is initialized as a random ordering of s distinct rules in the rule set (we used a chromosome size equal to the size of the rule set in our experiments: 50 for the original rule set and 72 for the new rule set, although smaller chromosome sizes were tested with no significant impact in score or algorithm performance). For each of the G generations, the fitness function is the result of playing n games for each game size and each pairing in mirror or mixed evaluation as described in section III.

Table II summarizes the parameters of our genetic algorithm.

IV. EXPERIMENTS AND RESULTS

In this section, we describe the experiments performed. All scores reported are obtained by averaging the score of an equal number of games with each possible game size (2 to 5 players), except for the specialized agents, who either play only two-player games or only 3-5 player games. All games are played with the standard hand sizes: 5 cards per player with 2 or 3 players and 4 cards per player with 4 or 5 players.

A. Validation results

Our first step was to reproduce the agents described in [11]. We implemented the seven rule-based agents described in subsection III-B. Note that the original article implements other agents, such as MCTS variations, but the well-performing ones do not outperform the best rule-based agent (Piers) and require a precise model of the other players, which is not available to us.

In the original paper, Walton-Rivers uses those seven agents as his test pool, and six of them (all except internal) are evaluated against this test pool. Our validation experiment consists of pairing the six agents with the seven agents in the pool. Table III shows the results of our validation, with most agents getting very similar results to the score reported by the original author, leading us to believe our implementation of the test pool is valid, which is an important initial step for the mixed evaluation experiments below.

B. Evolution using the existing rule set

For this experiment, we ran the evolutionary algorithm (in both mixed and mirror mode) using only the rules native to the framework, plus the conditional rules necessary to run the agents in the test pool. These are marked as categories 1, 2 and 3 in table IX. Since most of there rules (except category 3) correspond to rules described in [11], our objective for this experiment was to verify if, by throwing away any human assumptions of which order rules should be applied, but using a very similar rule set as the original article, we could get agents that outperformed the hand-crafted agents, in particular agent Piers, which is the best agent described in [11].

Because fitness is calculated by a number of simulations using random seeds, the fitness of an agent can fluctuate with each generation, even if the chromosome is unchanged (due to elitism). See figure 1 as an illustration of this fact. For that reason, after running the algorithm for 500 generations we took the agents corresponding to the 10 best performing chromosomes and ran a second round of simulations. The best mirror and mixed agent in this second round of simulations are referred to as MirrorOld and MixedOld in tables V and VI, which also shows their performance in all game sizes, the number of games played per agent and the standard error of the mean. We manage to beat the baseline on both mirror (from 18.16 to 18.38) and mixed (11.18 to 11.45) modes.

Table IV shows the best evolved chromosome for mixed play using only these "old" rules. Note that the first two rules are redundant, as if a card is more than 80% likely to be played, it is also more than 60% likely to be playable, but would not be redundant if there was another rule between them.

For space considerations, we do not show here the best chromosomes for the other categories, but we will make them publicly available in our repository after the competition[2].

C. New rules

After establishing that the existing rule set allows us to build better agents than our baseline, we attempted to extend the rule

[2]https://github.com/rubenrtorrado/hanabi

TABLE IV
CHROMOSOME FOR MIXED PLAY USING ONLY "OLD" RULES

Rule name
IF(Lives>1) PlayProbablySafeCard(0.8)
IF(Lives>1)PlayProbablySafeCard(0.6)
TellAnyoneAboutUsefulCard
PlaySafeCard
DiscardProbablyUselessCard(0.4)
DiscardUnidentifiedCard
DiscardOldestFirst
TellDispensable
TellRandomly

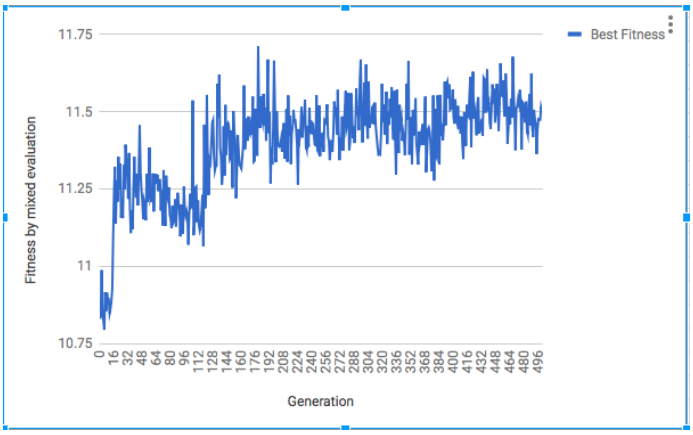


Fig. 1. An illustration of how fitness of the best agent varied per generation in one of our evolutionary runs for mixed mode with old rules

set, searching for rules that implement behaviors that are not covered by the existing rules. The first rule we added was **PlayJustHinted**. This rule will attempt to play a card that was just hinted by another player, accounting for their likely intentions. For example, if a player points to a card that you just drew telling it is a (5), when a (4) has just been played, it is likely that the hint was motivated by the fact that it is the correct color. PlayJustHinted scans the event history for all hints received since our last action, and attempts to play the most likely playable card among those that were pointed at by a hint. Optionally, we can choose to play a card only if it was pointed as a standalone hint (a hint that tells of no other cards), only if it is also our most recently drawn card, only if the probability of it being playable being $> p$, only if the number of lives is $> n$ or any combination of the above criteria. In the rule set, we included all combinations of probabilities p in $\{0, 0.2, 0.4, 0.6, 0.8\}$ and lives n in $\{0, 1\}$ with and without the restriction of standalone card or newest card .

The other rules we added attempt to give hints in a clearer, less ambiguous way to our partners. Sometimes, out of multiple playable cards, we need to decide which to hint and whether to tell color or value. Given that many successful agents take risks in playing cards that are not 100% sure to be playable, it is important not to give a hint that has the side effect of making a player believe an actually unplayable card to be playable. For example, a hint that points to three cards with value (2), when only one (1) has been played, leaves the player guessing which of them is the correct color. It might have been better to tell the color of the correct (2), especially if it happens to be the only card on its color. Conversely, if all (1)s had already been played, it is probably better to tell about all (2)s than hinting each color individually. The new rule **TellUnambiguous1** attempts to give information about a playable card by either maximizing the number of playable cards pointed to or minimizing the number of unplayable cards pointing to.

TellUnambiguous2, instead of looking at the number of pointed cards, calculates, from the other players perspective, the probability they would assign to each of their cards being playable. We do this for each possible hint. Each playable card is rewarded with a certain weight $w1 * p$, where p is the probability that it is playable from the other player's perspective. Each unplayable card is penalized with a factor of $w2 * p$. We select the hint that maximizes the value of a players hand in this way.

As before, we ran the evolutionary algorithm for 500 generations, then ran a secondary evaluation of the 10 best chromosomes. The resulting best agent is referred to as MirrorNew and MixedNew in tables V and VI. We improve the results compared to the agents evolved from the "old" rule set from 18.38 to 19.07 (mirror) and 11.45 to 11.65. We also noticed that our new rules appeared at high frequency at the start of the successful chromosomes. In particular, often many variations of PlayJustHinted coexisted at the start of the same chromosome. TellUnambiguous1 was often the highest priority "Tell" rule, while TellUnambigous2 was less successful.

D. Combining specialized agents

At this point, all of our agents are evolved by playing with game sizes from 2 to 5. We theorized that the strategy for playing when there are two players can be different from when there are five players. To prove this, we evolved a set of agents specialized in 2-player games and another set specialized in 3 or more-player games. After running a secondary evaluation on the two sets of specialized chromosomes, and finding the best specialized agent for game size of 2 and the best for game size of 3+, we built a situational agent that is a combination of both. It will check the number of players, then change its behavior to the one most suited for that game size.

We then ran an additional evaluation of the situational agent, which increased our scores from 19.07 to 19.32 (mirror) and from 11.50 to 11.65 (mixed), as shown in tables V and VI

V. ANALYSIS OF THE EVOLVED CHROMOSOMES

In this section, we look at the composition of the top 10 evolved chromosomes in each experiment in order to attempt to identify which rules were most successful and any discernible patterns that emerge in different experiments that could help explain why some strategies might work better for mirror or mixed play, or for specific game sizes. While we provide the frequency in which we observe some features in our most evolved chromosomes, we do not claim any statistical rigor to this analysis. Our findings serve only as illustration

TABLE V
RESULTS FOR MIRROR - NUMBER OF GAMES = 4*2000 = 8000 PER AGENT EVALUATED. BASELINE IS 18.16

Agent name	avg	2P	3P	4P	5P	s.e.m
MirrorOld	18.38	19.35	18.95	18.22	17.02	0.02
MirrorNew	19.07	19.61	19.68	19.11	17.87	0.02
MirrorSituational	19.32	20.07	19.58	19.36	18.29	0.02

TABLE VI
RESULTS FOR MIXED - NUMBER OF GAMES = 4*7*2000 = 56000 PER AGENT EVALUATED. BASELINE IS 11.18

Agent name	avg	2P	3P	4P	5P	s.e.m
MixedOld	11.45	12.13	12.05	11.08	10.55	0.02
MixedNew	11.50	12.11	12.19	11.15	10.56	0.02
MixedSituational	11.65	12.38	12.29	11.22	10.72	0.02

of our results and determining their validity could serve as a future research question.

For the purpose of this section, we analyze only the runs including the new rules. We picked the top 10 chromosomes from MirrorNew and MixedNew (unspecialized), the top 10 chromosomes specialized at 2 player games in MirrorSituational and MixedSituational and the top 10 specialized crhomosomes for games with 3 or more players in MirrorSituational and Mixed situational.

First, we examine the success of our new rules. In table VII, column (I), we see how often some variation of **PlayJustHinted** was selected as the very first rule in a chromosome. In total, this happens 24 out of 30 times in mirror mode, but only 13 out of 30 times in mixed mode. This could be due to the fact that agents such as Legal Random and Flawed in the test pool do not favor giving playable hints over giving random hints. Thus, in mixed mode, assuming a card is playable just because it was hinted is a less reliable strategy.

Because most agents have a rule for playing a card as highest priority, TellUnambiguous was not as often chosen as first rule of the chromosome. Nevertheless, it was very often chosen as first among the tell rules (rules that result in giving a hint). In column (II) of table VII, we see how often this happens for each game mode. Overall, it was the preferred tell rule 19 out of 30 times in mirror mode, and only 7 out of 30 times in mixed mode, most often in the variation TellUnambiguous1. This rule attempts to maximize the number of playable cards pointed to with each hint, or minimize the number of unplayable cards. As such, it pairs extremely well with PlayJustHinted, which is less popular in mixed mode and could explain the relative lack of success of TellUnambiguous in this scenario. A popular tell rule for mixed mode was CompleteTellUsefulCard, which gives complete information about a useful card and could be very effective with the agents in the test pool that require complete information for playing a card.

The last part of our analysis is regarding the difference in strategy from 2 player games to games with 3 or more players. Table VIII shows how often a play or tell rule was selected as first of its chromosome for the specialized 2 player or 3+ player games. In games with many players, playing a card nearly always takes precedence over giving a hint, with 19 out of the 20 chromosomes following this pattern. In 2 player games, there is a fairly even split between agents that prioritize playing over telling (12 out of 20) and telling over playing (8 out of 20). We suspect that in 2 player games, it is often correct to hold a card you know is playable in order to give a hint to your partner, to avoid them accidentally discard a useful card, for example. In games with more players, players who already know a playable card should probably give higher priority to playing it, and leave the task of giving hints to other players. By choosing to give a hint instead of playing your card, you would consume a hint that could better be utilized by a player who knows little of their own hand (and would be forced to discard if they got to their turn without an available hint token).

TABLE VII
PREVALENCE OF NEW RULES BY EXPERIMENT

Game Type	PlayJustHinted as first in chromosome (I)	TellUnambiguous as first tell (II)
Mirror 2 players	5/10	7/10
Mirror 3+ players	10/10	10/10
Mirror unspecialized	9/10	2/10
Mirror overall	**24/30**	**19/30**
Mixed 2 players	4/10	6/10
Mixed 3+ players	4/10	1/10
Mixed unspecialized	5/10	0/10
Mixed overall	**13/30**	**7/30**

TABLE VIII
PRIORITY OF PLAY AND TELL RULES BY GAME SIZE

Game Type	Play before Tell	Tell before Play
Mirror 2 players	7/10	3/10
Mixed 2 players	5/10	5/10
2 players overall	**12/20**	**8/20**
Mirror 3+ players	10/10	0/10
Mixed 3+ player	9/10	1/10
3+ players overall	**19/20**	**1/20**

VI. FUTURE WORK

As future work, other than implementing other well-performing rules, which requires human expertise, it would be good to have a language of primitives from which the rules themselves could be evolved. Still in the topic of evolution, rather than have a list of rules that must be examined in

order, a Neural Network with evolved weights could determine which rule (or action) to take, similarly to the approach of [18]. Alternatively, such controller could be developed by techniques such as Deep Reinforcement Learning [19].

Another important development would be the ability to generate a model of the other players' behavior, such as how risk-taking they are when playing a card, their preferred discard policy, etc. If we could accurately recognize these features during game play, we could evolve specific chromosomes for playing with agents with those characteristics and so improve our mixed score. We are particularly interested in initiatives such as [20], which attempts to build a model of unknown partners during cooperative gameplay by interpolating between known models. These models can also be used for non-rule-based agents, such as MCTS [12], which require a model of the other players.

VII. CONCLUSION

We used evolution in three steps to get better Hanabi-playing agents than the human-created baselines: First we evolved the order in which rules are applied, using a set of rules very similar to the ones used in [11]. As the quality of an agent depends not only on the ordering of the rules, but also on the expressiveness of the rule set, we then added rules that account for our partner's intentions (assuming a hinted card has high probability of being playable) and to choose which piece of information to give about a playable card in the least ambiguous way possible. This not only brought a quantitative increase to our score, but we also noticed qualitatively that the new rules were in general very effective, appearing at the head of many of the most successful chromosomes.

Finally, we created specialized agents for specific game sizes and using their behavior for any game size, we get an improvement over a generic agent that is optimized for playing all game sizes. This shows that the best strategy for Hanabi likely depends on the number of player. We analyzed 30 of our best chromosomes to attempt to identify patterns that make some strategies better in each game size, and also for mirror or mixed evaluation.

By combining evolution, new rules and specialized behavior, we get a improvement over the best purely rule-based agents, going from 18.16 to 19.32. While hat agents [13], [14] score significantly better than our mirror agents, they are unsuited for mixed or human play. To our knowledge, the only published non-hat agent that exceeds our score is the combination of Tree Search with a rule-based agent as evaluator, seen in [14], with a score of 20.22 across all game sizes. However, that agent vastly exceeds the time budget allowed in the competition. It is also worth noting that, as future work, we could attempt to combine our own rule-based agent with Tree Search algorithms, or even use rule-based agents evolved specifically for this purpose.

In mixed mode, our improvements were smaller, but still expressive, going from 11.18 to 11.65. It is important to note that the test pool used for the mixed evaluation consists on some agents, such as Flawed and Random that are purposely very bad (scoring around 5 points in average), so scores in this mode are naturally expected to be lower. While we do not know which agents will be used as the competition test pool, we hope our improvements when pairing with the test pool of [11] translates to good results in the mixed competition.

ACKNOWLEDGMENT

Rodrigo Canaan gratefully acknowledges the financial support from Honda Research Institute Europe (HRI-EU).

REFERENCES

[1] J. Schaeffer, R. Lake, P. Lu, and M. Bryant, "Chinook the world man-machine checkers champion," *AI Magazine*, vol. 17, no. 1, p. 21, 1996.
[2] M. Campbell, A. J. Hoane Jr, and F.-h. Hsu, "Deep blue," *Artificial intelligence*, vol. 134, no. 1-2, pp. 57–83, 2002.
[3] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, pp. 484–489, 2016.
[4] BoardGameGeek, "Hanabi," access: 05/14/2018. [Online]. Available: https://boardgamegeek.com/boardgame/98778/hanabi
[5] H. Osawa, "Solving hanabi: Estimating hands by opponent's actions in cooperative game with incomplete information." in *AAAI workshop: Computer Poker and Imperfect Information*, 2015, pp. 37–43.
[6] Spiegel, "Das sind die ausgezeichneten spiele des jahres," access: 05/14/2018. [Online]. Available: www.spiegel.de/netzwelt/games/spiel-des-jahres-2013-das-sind-die-ausgezeichneten-brettspiele-a-910001.html
[7] J. Walton-Rivers, "Fireworks agent competition," access: 05/14/2018. [Online]. Available: hanabi.aiclash.com
[8] J.-F. Baffier, M.-K. Chiu, Y. Diez, M. Korman, and V. Mitsou, "Hanabi is np-complete, even for cheaters who look at their cards," 2016.
[9] M. J. van den Bergh, A. Hommelberg, W. A. Kosters, and F. M. Spieksma, "Aspects of the cooperative card game hanabi," in *Benelux Conference on Artificial Intelligence*. Springer, 2016, pp. 93–105.
[10] M. Eger, C. Martens, and M. A. Córdoba, "An intentional ai for hanabi," in *Computational Intelligence and Games (CIG), 2017 IEEE Conference on*. IEEE, 2017, pp. 68–75.
[11] J. Walton-Rivers, P. R. Williams, R. Bartle, D. Perez-Liebana, and S. M. Lucas, "Evaluating and modelling hanabi-playing agents," in *Evolutionary Computation (CEC), 2017 IEEE Congress on*. IEEE, 2017, pp. 1382–1389.
[12] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of monte carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in games*, vol. 4, no. 1, pp. 1–43, 2012.
[13] C. Cox, J. De Silva, P. Deorsey, F. H. Kenter, T. Retter, and J. Tobin, "How to make the perfect fireworks display: Two strategies for hanabi," *Mathematics Magazine*, vol. 88, no. 5, pp. 323–336, 2015.
[14] B. Bouzy, "Playing hanabi near-optimally," in *Advances in Computer Games*. Springer, 2017, pp. 51–62.
[15] E. J. Anderson and M. C. Ferris, "Genetic algorithms for combinatorial optimization: the assemble line balancing problem," *ORSA Journal on Computing*, vol. 6, no. 2, pp. 161–173, 1994.
[16] A. Moraglio, J. Togelius, and S. Lucas, "Product geometric crossover for the sudoku puzzle," in *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*. IEEE, 2006, pp. 470–476.
[17] B. L. Miller, D. E. Goldberg *et al.*, "Genetic algorithms, tournament selection, and the effects of noise," *Complex systems*, vol. 9, no. 3, pp. 193–212, 1995.
[18] C. Holmgård, A. Liapis, J. Togelius, and G. N. Yannakakis, "Evolving personas for player decision modeling," in *Computational Intelligence and Games (CIG), 2014 IEEE Conference on*. IEEE, 2014, pp. 1–8.
[19] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.
[20] S. Barrett, P. Stone, and S. Kraus, "Empirical evaluation of ad hoc teamwork in the pursuit domain," in *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*. International Foundation for Autonomous Agents and Multiagent Systems, 2011, pp. 567–574.

TABLE IX
APPENDIX - RULE SET

Rule name	Description	Category
Play If Certain	Plays a card with fully known information that is playable	1
Play Safe Card	Plays a card that is known to be playable (even with partial information)	1
Play probably safe card (p)	Plays card most likely to be playable if the probability of being playable is greater than p	1 [1]
If lives >1 play probably safe card (p)	Plays card most likely to be playable if the probability of being playable is greater than p and there is more than one available life	2 [2]
If Hail Mary play probably safe card (p)	Similar to above, but also requires the deck to be empty	2 [3]
Complete Tell Useful Card	Tells the missing information (color or value) of a partially known playable card to a player	1
Tell About Ones	Tells a player about all their cards with value of one	1
Tell Anyone About Useful Card	Tells a player some new information about one of their playable cards, prioritizing value if card is completely unknown.	1
Tell Anyone About Oldest Useful Card	Tells a player some new information about their oldest playable card, prioritizing value.	1
Tell Playable Card	Tells a player some information about a playable card. Decides randomly between value and color, even if the card is partially known	1
Tell Playable Card Outer	Same as Tell Anyone About Useful Card	1
Tell Anyone About Useless Card	Tells a player some information about a card that is never going to be playable	1
Tell Dispensable	Same as Tell Anyone About Useless Card	1
Tell Fives	Tells a player about all their cards with value five	1
Tell Most Information	Gives a hint that tells the most information, or most new information about their hand	1 [4]
Tell Randomly	Gives a random hint to a player	1
Tell Unknown	Gives new information about a card in a player's hand.	1
Discard Useless	Discards cards whose pre-requisites have been discarded.	1
Discard Safe	Discards that is no longer playable.	1
Osawa Discard	Discards a card that is useless or safe.	1
Discard If Certain	Discards a card that with fully known information and no longer playable	1
Discard Highest	Discards card in hand with highest known value.	1
Discard Oldest	Discards oldest card in hand.	1
Discard Oldet No Info First	Discards oldest card with no known information	1
Discard Unidentified Card	Discards a card with no known information	1
Discard least likely to be necessary	Discards card with smallest probability of being necessary for a perfect score	1
Discard probably useless (p)	Discards card with highest probability of being useless, as long as that probability is greater than p	1 [2]
Play Finesse	Part of the Finesse strategy.	3
Play Finesse Told	Part of the Finesse strategy.	3
Tell Finesse	Part of the Finesse strategy.	3
Play Unique Possible Card	Part of the Finesse strategy	3
Tell Illinformed	If a player is ill-informed, give them a hint.	3
Try To Unblock	If there is no unblocking player between you and a blocked one, unblock.	3
Legal Random	Selects a random legal action	1
PlayJustHinted	See section IV	4
TellUnambiguous1	See section IV	4
TellUnambiguous2	See section IV	4

Categories:
1: Already implemented in the framework and similar to a rule described in [11]
2: Implemented using the framework's IF and/or CONDITIONAL rule and described in [11]
3: Implemented in the framework, but not described in [11]
4: New rules implemented for this work

Parameters:
[1] - Values of p = 0, 0.2, 0.25, 0.4, 0.6 and 0.8 were used for this rule
[2] - Values of p = 0, 0.2, 0.4, 0.6 and 0.8 were used for this rule
[3] - Values of 0 and 0.1 were used for this rule
[4] - Only the variant that tells the most information was used for this rule

Standard Economic Models in Nonstandard Settings – StarCraft: Brood War

Bryan S. Weber
Department of Economics
College of Staten Island: CUNY
2800 Victory Blvd.
Staten Island, NY 10314
bryan.weber@csi.cuny.edu

***Abstract*—This paper reviews an entrant to this year's StarCraft: Brood War AI tournament, CUNYbot. CUNYbot makes strategic decisions using a low-dimensional economic model traditionally used to describe the behavior of countries, but has applications for any real-time-strategy game (RTS) where the capital/labor ratio (k) and technology/labor ratio (t) are critical. CUNYbot first tunes the economic model parameters between games using a genetic algorithm, allowing it to learn an optimal static k for each built-in AI race. In the second step of the project, CUNYbot models the opponent during the game and is permitted to react to the opponent's inferred choices in k. The reactive model adopts a greedy "tit-for-tat" style strategy against all three races. This paper reviews the features of the capital-augmenting Cobb-Douglas model, the game theory behind the reactive strategy, and demonstrates adaptation to particular opponents in the RTS context.**

***Index Terms*—StarCraft, genetic algorithms, economic modeling**

I. Introduction

In this paper I explore an exotic application of a well-used economic model, the Cobb-Douglas (CD) model, which dates back to 1928. [1], [2], [3] This model is applied to the modern video game StarCraft: Brood War released 70 years later. StarCraft: Brood War is played by managing workers and a starting resource depot from one of three races, Terran, Protoss, and Zerg. These workers collect resources, build infrastructure, and eventually lead to the construction of combat units. The winner of the game is the last one with a standing structure.

To approach this game, I first use the historic CD model as a heuristic utility function for CUNYbot, which allows for the derivation of an optimal army/worker and technology/worker ratio for CUNYbot (k and t, respectively). When training CUNYbot, instead of refining the build orders or unit- and time-specific reactions, here, I only train the CD parameters which define k and t. [4] After training the economic parameters using a genetic algorithm, the bot identifies a significantly different ratio of k and t for each of the three built-in opponents.

Second, I use the observed game information to evaluate a CD model of my opponents' expenditures in real time. CUNYbot is then retrained with the opportunity to respond to the current model of its opponent. Giving CUNYbot responsiveness leads to CUNYbot adopting a tit-for-tat style strategy against all three races. This is indicated by CUNYbot playing a passive, worker-intensive (greedy) approach whenever possible, but reacts to the opponent's aggressive behavior.

Both of these points suggest that the historic Cobb-Douglas model continues to be useful describing economies in the RTS setting. The remainder of this paper continues with a review of relevant literature, and a discussion of CUNYbot and the CD model. I then discuss the genetic algorithm used for tuning CD parameters. Next, I explore the final results of training the static CD model against each of the three built-in AIs. Then, I show how to infer CD parameters for the opponent, demonstrate that reactive CUNYbot plays a tit-for-tat style strategy, and conclude.

II. Literature Review

Previous work on SC:BW emphasizes either the detailed arrangement and targeting choices of individual troops, *micro*, the management of the economy within the game and purchasing decisions, *macro*. These decisions are complicated in StarCraft: Brood War because of 3 primary features:

1) Strategic Interaction: The actions of both players directly affects the payoffs of others.
2) Massive Action Space: StarCraft is played on a game board several thousand pixels wide and high. Players frequently manage 200 or more units at a time. These units have dozens of distinct classes. By comparison, chess manages 8 classes of pieces on an 8 by 8 board.
3) Limited Information: The actions of the opponent are heavily obscured by the *fog of war*, an obscuring mist over most of the map, and only hints of their overall plan are available.

This work primarily explores macro, the formal macroeconomics of RTS games, and the strategic interaction between players. It is tempting to assume that economics will relate entirely to the collection of resources, and indeed some fundamental work has been done purely on improving mining effectiveness. [5] This technique, mineral-locking, has already been incorporated into CUNYbot. However, it is important to be clear that the emphasis of the CD model is scarce resource management, not resource collection. Economic models operate by evaluating a utility function, and identifying the opportunity costs for each decision. [6]

978-1-5386-4359-4/18/$31.00 ©2018 IEEE

Economics has a long history of studying games to derive an optimal strategy or an equilibrium. But these economic games are generally far more streamlined than SC:BW. While not an exhaustive list, the literature contains extensive study of the following game types: Prisoner's Dilemma (where tit-for-tat derives), Cournot Production game, and a variety of location games. [7], [8], [9], [10], [11] These games have strategies that involve either very few decisions for each player, or repeated rounds of the same decisions. More complex games are generally not suitable for evaluating equilibrium, and therefore this type of economic literature is sparse. In games like chess, the executional complexity leads to a distracting emphasis on improving execution, rather than the pursuit of an objective best solution. [12], [13]

While economists have limited ability to solve extremely complex games, this does not prevent economists from studying their virtual economies, such as those contained within *Everquest* or *World of Warcraft*. [14], [15], [15], [16] These virtual worlds have human participants who have served as experiment subjects. [17] One researcher has already used the Cobb-Douglas model to describe the economy of Diablo 3, highlighting the prevalence of this specific model. [18] Standard macroeconomic models like Cobb-Douglas have frequently been used to describe existing economies, but have not been used to participate. In the bot participants of SC:BW, all utility functions or economic reasoning I found was uniquely designed or trained. To my knowledge, this is the first time a historically established macroeconomic model has been used to create a participant.

Much of the prior computer science work on macro focuses on build orders, ie. choosing an opening sequence for units. [19], [20] For most bots in the competition, this takes the form of either a single fixed build order, or a choice among several fixed build orders, usually taken from expert knowledge. There is much work to be done in this field. ZZZKbot, for example, was noted for exclusively using hard-coded builds, often targeted for specific opponents. [21] When uncertain about which opening to use, many bots now use reinforcement learning techniques. [21] Some bots have been trained to select build orders using genetic algorithms, even with an eye towards economic victory. [22]

This approach of a fixed build order inherently is unable to react to unorthodox decisions by the opponent. Not all bots use explicitly hard-coded build orders, however. More adaptive approaches have been made using case-based reasoning, deep learning, and replay mining to determine an optimal strategy. [23], [24], [25], [26] Others have used genetic algorithms to find the shortest time for a particular unit composition [27], or to assign unit tasks. [4]

I also use a genetic algorithm approach to evaluate the strategy. My search is explicitly for the parameters of the CD model, with an interest in the implied parameters k and t, rather than build orders or individual unit tasks. [4], [22], [28] The choice between workers or army composition after the opening game has been directly explored by Oh, *et al*. [29] While other bots have used customized economic models in the past, CUNYbot is the only bot I have found that incorporates any of the standard economic models into its strategic choice on what and when to buy.

III. About CUNYbot

CUNYbot (City University of New York - Bot) is an AI that plays a complete game of StarCraft: Brood War as the Zerg race. It is a new bot built using BWAPI 4.2.0, a project of Adam Heinermann. [30] CUNYbot can be regularly observed playing against other bots on the Student StarCraft AI Tournament (SSCAIT), and participated on the StarCraft AI Ladder (SAIL). Its rank has fluctuated between 1700 to 1900 on the SSCAIT ladder. CUNYbot is, at its heart, a state machine, where the three primary states are determined by the CD model of country growth. The CD model provides a heuristic utility function. From this, CUNYbot determines priorities for its limited resources, whether that is troops, workers, or technologies. I discuss this model in great detail below.

CUNYbot is almost exclusively a platform for testing the CD model. The other systems, such as those regarding micromanagement or building choice, are rule-based and do not evolve between games. Changes in other systems would interfere with the training of the CD model. I give a brief overview of the systems that govern popular research topics in SC:BW below.

1) Map Analysis (*Inventory*): A complete grid of 8x8 minitiles is recorded on game start, as well as the possible starting locations of all players and initial map resources. Two maps are stored, both made by fire-filling the walkable tiles. The first map counts out from CUNYbot's center of base. The second counts out from a suspected enemy base, determined by the start location or center of buildings. Both maps are updated when new buildings are found or destroyed.
2) Micromanagement (*Fight_MovementManager*): If CUNYbot is in the army-starved state, all troops are attracted to its home base. Otherwise, they are attracted to the enemy base. This direction is easily determined by higher or lower values in the generated maps. At every frame, each CUNYbot unit evaluates the value of friendly units in the neighborhood and the nearest neighborhood of enemy units. This valuation is done by unit cost, a metric with precedence in the literature. [31] If the value of friendly units exceeds that of opponent units, CUNYbot attacks. It targets enemy units using a heuristic of distance and target characteristics (eg. nearby units with spells first, distant buildings last). Lastly, CUNYbot will attack everything without considering retreat if its army is three times larger than its opponent's.
3) Unit Choices (*AssemblyManager*): If CUNYbot is in the labor-starved or the technology-starved state, it builds workers. Otherwise, it is army-starved and builds combat units. The choice in which units to build is based off of a heuristic analysis of the opponent's composition.

CUNYbot does not build any units that cast spells, such as Queens or Defilers.

4) Building Choices (*TechManager*): If CUNYbot is in the technology-starved state, it builds more advanced technology buildings. It emphasizes air if the enemy lacks air defense (by cost), and ground otherwise. If it is labor-starved, it builds additional expansions. If it is army-starved, it builds additional production facilities, or static defenses. Exceptions: If the opponent has military forces nearby, and CUNYbot has inadequate defenses, it may reflexively build anti-air technology or static defenses. If the game is just beginning, CUNYbot executes a recorded build order until a worker or building is destroyed.
5) Learning Between Games (*GeneticHistoryManager*): This records and writes important information: CD model, build orders, and any other parameters, before and after each game.
6) Collection of Minerals (*MiningManager*): This module keeps track of workers, assigns them to collect minerals or gas, and assigns workers to clear obstructing mineral patches when needed.
7) Cobb-Douglas Model (*Cobbdouglas*): This governs the three states: army-starved, worker-starved, or technology-starved. The model is explained in great detail in Section IV.

At the end of each game, CUNYbot takes note of its build order, the map, the opponent, the CD model parameters, and a convergence rate r using *GeneticHistoryManager*. In this paper CUNYbot is restricted to only train against one opponent, on one map, using the same build order. All other parameters are fixed between games. CUNYbot explicitly does not make any strategic reactions to the opponent's race.[1] This leaves CUNYbot training exclusively on CD parameters and the convergence rate r.

IV. The Cobb-Douglas Model

The Cobb-Douglas (CD) model is a popular parametric model primarily used to describe the growth rates of countries. It is simple enough that without knowledge of its effectiveness in other contexts, it might easily be overlooked or neglected in modeling RTS behavior. It divides resources into a few fundamental classes which I describe in context below:

- Capital (K): The total value of tools that directly help win the game. ex: Hydralisks, or Terran Marines. For the purposes of this paper, army units.
- Labor (L): The total value of workers themselves, ex. Drones, Probes, or SCV's.
- Technology (T): The total value of items which augment the capital such as research, upgrades, ex: Ranged Missile Attacks +1, Hydralisk Den.
- Gross Domestic Product (Y): The total output of goods and services, utility. The value function of the player.

[1]CUNYbot, for example, does not change its scouting habits, unit composition, or building choices against particular races. Its strategies remain unchanged.

This categorization is particularly clear for StarCraft's Zerg race - whose technological structures T permit construction of new capital K but do not themselves produce it. As such CUNYbot was designed to play Zerg. In this model, technology is incorporated as capital-augmenting because technology directly improves the value of the capital. Conversely, technology does not have an immediate effect on the productivity of labor. [3] The functional form for capital-augmenting CD is:

$$\begin{aligned} &a)\ Y_f = (A_f K_f)^{\alpha_{K,f}} L_f^{\alpha_{L,f}} \\ &b)\ \alpha_{L,f} = 1 - \alpha_{K,f} \\ &c)\ A_f = T_f^{\alpha_{T,f}} \end{aligned} \tag{1}$$

Where f is the count of frames in the game. The normalization in 1b loses no generality because Y is an ordinal utility function. 1c is assumed because technical growth will not be time-dependent, but instead will be chosen by the player. To begin, I assume all three α parameters are static throughout the game in Section V-A, and consider the flexible values of α later in Section V-B. In order to determine the value of my current stock of K_f, L_f and T_f, I use the stock value of the unit or item:

$$mineral\_cost + 1.25 * gas\_cost + 50 * supply\_cost \tag{2}$$

This straightforward metric was used as a baseline measure of the value of a StarCraft force by Synnaeve and Bessière. It was sufficient to predict the winner of combat 63% of the time for similar forces. [31][2] There may be more precise conversions to evaluate total stock, but this maintains a clear connection to economic work, and has precedent in research.[3] For combat units that can be damaged, I multiply their stock value by their proportion of health remaining. Beyond combat damage, there is no explicit depreciation in SC:BW. In fact, the opposite occurs, most units heal over time if left alone.

At every frame f, the agent chooses how best to allocate its limited resources to purchase additional stock of K, L and T. The bots' three states are then respectively capital-starved (army-starved), labor-starved, and technology-starved. For each of CUNYbot's three states, the bot invests in the asset type which has the largest gradient. This gradient measures the expected benefit of each type of investment. The opportunity cost (defined as the value of the next best foregone option) for any investment in K_f, L_f and T_f is the next largest

[2]For upgrades and researches, the supply cost is 0. For Zerg buildings, which consume a worker, I include the cost of the worker. I do not have a metric for the cost of the larva.

[3]CUNYbot also assesses combat using this metric. If a local enemy's force is larger than friendly forces by this metric, it generally avoids attack.

gradient.[45] It is a trivial matter to find the gradients of the CD function once given the parameters $\alpha_K, \alpha_L, \alpha_T$. In an equilibrium CUNYbot keeps those gradients equal and minimized, allowing us to evaluate the ratios $k = K/L$ and $t = T/L$.[6] They are:

$$\begin{aligned} k &= \frac{\alpha_K}{\alpha_L} \\ t &= \alpha_T k = \frac{\alpha_K \alpha_T}{\alpha_L} \end{aligned} \tag{3}$$

In historical economic research, these ratios were assumed to be relatively unchanged throughout the lifetime of the economy. In practical application, these ratios have remained consistent for countries like the United States. [1] This simple approach was also modestly successful for earlier versions of CUNYbot in the SSCAIT 2017 challenge. I loosen this assumption in Section V-B. Figure 1, shown below, illustrates the path of CUNYbots's economy Y_f over time for a given starting set of $\alpha_K, \alpha_L, \alpha_T$ against StarCraft's built-in Zerg opponent. The value of Y_f was measured every in-game second (24 frames).

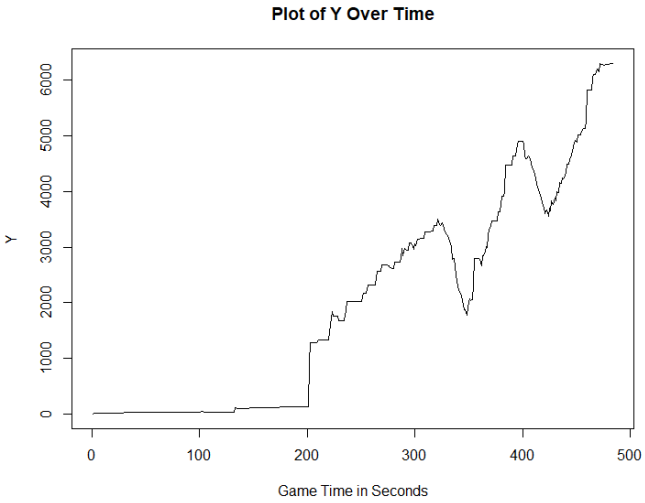


Fig. 1. CUNYbot's economy over time winning against a built-in Zerg AI. Parameters were: $\alpha_K = 0.46, \alpha_L = 0.54, \alpha_T = 0.39$.

[4]I note that for Zerg, production of both capital (K) and labor (L) are mutually exclusive. New Zerg units of either type are produced through larva, which is exhausted on use. Meanwhile, production of technology is done either by expending workers L or through technology buildings that would otherwise be unoccupied. As a result, expenditures on K and L do not entirely exclude the investment in T, which will be purchased if there are left over resources.

[5]CUNYbot also seeks the enemy when not capital-starved. It has "enough" army. However, the ultimate act of attacking will only happen if the bot assesses itself as winning the fight, or its army gradient is 0, ex. it is at 200 supply and cannot build a larger army).

[6]There are also several discontinuities in StarCraft itself that are not accounted for in the purely continuous CD model. For example, at the beginning of the game, one cannot purchase military units (K) directly, and at the end of the game, supply cannot increase beyond 200. These edge conditions are accounted for by setting the gradient of the respective stock to 0.

The first 200 seconds represent the opening book period where only workers are built. The CUNYbot performed a standard 12-hatch, after which it has no army and enters an army-starved state. When CUNYbot leaves the army-starved state, it moves its army towards the opponent and begins to fight. The first fight is clearly visible around 300 seconds, and another around 400 seconds. In the first fight, players exchanged units and typical growth was inhibited. From 350-400 CUNYbot rebuilt its army. It attacked again at around the 400 mark. The built-in Zerg opponent resisted, but did not successfully defend. The game ends shortly thereafter as the base AI's buildings are destroyed.

In sum, the CD model leaves us with a simple vector (genome) g that needs to be optimized: $g = \{\alpha_K, \alpha_L, \alpha_T\}$. These values allow the derivation of k and t. A RTS-specific interpretation of these parameters is easy to manage. I provide some stylized interpretation below:

1) Aggressive: $(\alpha_K >> \alpha_L)$ This style of play aims to end the game in the short run by buying a large army.
2) Safe: $(\alpha_K \approx \alpha_L)$ This style of play aims to grow, but with sufficient security to protect against aggression.
3) Greedy: $(\alpha_K << \alpha_L)$ This style of play aims to grow the income of the player, but purchases only a trivial or nonexistent defensive force.
4) Technical Play: (Large α_T) This style of play encompasses rushing to build special units to win the game. The main units CUNYbot can use of this type are Lurkers, which cannot be seen without the defending player also possessing special units.

I note that these are relative interpretations of RTS game strategies. One could evaluate the CD model to determine if a player was more safe or more aggressive. But, there is no explicit threshold where an opening is "safe" or "aggressive", they are relative to each other. Below, I outline the setup of the experiment, and show the genetic algorithm used to train CUNYbot's style against the built-in AI.

V. Methodology & Results

A. Training the Static Cobb Douglas Model

Training was done in parallel using 6 copies of *Chaoslauncher: MultiInstance* with game speed set to 0, the fastest possible setting. [32] This created 6 identical copies of CUNYbot. At the start of each game, CUNYbot reads the genome, reads win/loss information in *output.txt*, and executes the genetic algorithm outlined below. At the end of each game, CUNYbot appends the genome and game result to *output.txt*. As a result, the output file is in order of game completion.[7]

In order to prevent delays, CUNYbot resigns if the game goes on too long or is delayed, which happens between 1.5-0.2% of games depending on the opponent. Otherwise no time limit is imposed on the games and they are left alone.[8]

[7]A careful review of the file will reveal that other information is also written to *output.txt* (eg, map, opponent name, lags/delays), but it has been ignored for this exploration.

[8]System notes: AMD Ryzen 7 1700, 3Ghz processor, 16GB RAM, total training time is approximately 7 hours of wall time for 1000 games.

Algorithm 1 Genetic Algorithm (GA)

procedure TRAIN GA
  $P \leftarrow \emptyset$. ▷ Define Terms
  $g \leftarrow \{\alpha_K, \alpha_L, \alpha_T\} \sim iid.\ U(0,1)$. ▷ Begin Test
  $\alpha_K \leftarrow \frac{\alpha_K}{\alpha_K+\alpha_L}, \alpha_L \leftarrow \frac{\alpha_L}{\alpha_K+\alpha_L}$. ▷ Normalize
  CUNYbot plays training opponent.
  if CUNYbot wins **then** ▷ Keep Fit Genomes
    Add g to the top of P.
  else goto 3.
  while $P.size() > 50$ **do** ▷ Death
    Remove bottom (oldest) element of P.
  if $P.size() = 50$ **then** ▷ Geometric Crossover
    Randomly draw $\{g_{p1}, g_{p2}\} \in P$.
    $g_c \leftarrow g_{p1}^{1-a} g_{p2}^{a}$, where $a \sim iid.\ U(0,1)$.
  else goto 3.
  if $U(0,1) > 0.95$ **then** ▷ Mutate Child
    Randomly choose one element, $e_{old} \in g_c$.
    Replace $e_{old} \leftarrow e_{old} + m$, $m \sim iid.N(0, 0.05)$.
    if $e_{old} > 1$ **then** $e_{old} \leftarrow 1$.
    if $e_{old} < 0$ **then** $e_{old} \leftarrow 0$.
  else goto 4.
  if less than 1000 games played **then**
    goto 4.
  else end ▷ End Training

While in competition CUNYbot chooses from among various fixed openings. For this experiment the bot is locked into a single, short, hard-coded build order called a 12-hatch.[9] It is important that the bot is forbidden to change opening selection (or any other non-CD parameter) as I train on the CD model, since any improvement in win rate could be confounded. Beyond this fixed opening, the ratio of k and t are determined by the CD model above. The map chosen was *(4)Fighting Spirit.scx* because it is generally free of complicating obstacles that might otherwise obstruct the bot. The random seed provided to SC:BW was locked and unchanged for every game.

I also experimented with arithmetic crossover ($g_c \leftarrow g_{p1} * (1-a) + g_{p2} * a$), but the convergence was noticeably slower. It did not converge after even 2000 games. The parallel shape shared by both the objective function and the geometric crossover suggests geometric crossover is more appropriate, and it does lead to a much faster convergence. [33] As mentioned, the CD parameters change only between games through the genetic algorithm and no other features are altered. This process was costly in training time but was very robust in practice, since the child genome was composed of linear combinations along a continuum of known winning approaches. Convergence happened quickly, even though the setup was comparable to examples that take much longer over rough surfaces. [34] No special techniques other than geometric crossover were utilized and fitness was evaluated purely on the victory of the game. [4]

The results of training displayed in Figure II show the final generation of 50 genomes after playing 1000 games against each race of the built-in AI. CUNYbot converges to significantly different genomes depending on the race, suggesting that the CD model has practical merit in describing RTS economies.

TABLE I
STATIC CD LEARNING - FINAL 50 GENOMES WITH 1000 TRAINING ROUNDS

Mean	α_K	α_L	α_T	k	t
Static CD vs P	0.51	0.49	0.38	1.05	0.39
W:59.5%	(0.01)	(0.01)	(0.01)	(0.03)	(0.01)
Static CD vs T	0.55	0.45	0.57	1.21	0.70
W:76.9%	(0.00)	(0.00)	(0.01)	(0.02)	(0.02)
Static CD vs Z	0.47	0.53	0.36	0.87	0.31
W:100.0%	(0.01)	(0.01)	(0.00)	(0.02)	(0.01)

Note: SD of each genome element in parentheses. Win rate is evaluated starting from the time the first member of the final generation is added to the end of the experiment.

The parameters of the CD model allow for easy interpretation, and I have calculated the ratios of k and t directly. For example, the bot chooses its highest values of k and t against Terran, suggesting smaller economies while emphasizing well-upgraded armies. CUNYbot manifests this general preference most visibly through its Lurker upgrade, which takes advantage of the Terran's relatively limited ability to detect them. On the other hand, the CUNYbot chooses very low k and t against Zerg. In observation, CUNYbot expresses low k and t by expanding often and defending with relatively low-tech units, such as Zerglings. High-tech units that CUNYbot is familiar with, such as Lurkers and Ultralisks, are rarely seen in professional Zerg vs Zerg matchups.[10]

Against Terran and Protoss, CUNYbot learned to play aggressively compared to an untrained mean of about 0.5 for all parameters. The built in Zerg stands out as the only case where it did not play aggressively, but this built-in AI is also clearly the weakest against CUNYbot. Further training may yield more refinements, but the distinct convergence of CD parameters by race is clear, highlighting the bot's unique approach against each opponent.

B. Training the Reactive Cobb-Douglas Model

The above static approach has converged to a fixed genome g that remains constant within each game. It found distinct results for each of the three opponents, suggesting three distinct, but relatively aggressive styles. However, opponents do not have to maintain a constant g, nor do they need to maintain any sort of balanced investment pattern within a game. Indeed, such variations are central to RTS games in general. Instead of fixed values of $\alpha_K, \alpha_L, \alpha_T$, an opponent will face decisions

[9]This opening consists of building to twelve workers as soon as possible, then building a hatchery.

[10]I note that one advanced technical unit, the Mutalisk, is popular with human players in this match, but CUNYbot does not have comparable mastery over Mutalisks at this time. As the CUNYbot gets better at Mutalisk micro, I would anticipate α_T and t would increase.

of aggression or greed that repeat continuously throughout the game. They can be approximated to have their own values of $g_{opp,f} = \{\alpha'_{K,f}, \alpha'_{L,f}, \alpha'_T\}$ in each frame of the game. Their choices require a strategic and careful response by CUNYbot.

The *tit-for-tat* strategy is characterized by friendly play and mild retaliation. First, one plays a friendly and non-aggressive strategy. That is to say, it plays greedy. Second, if the opponent plays aggressively, we begin to play safe to punish this behavior. Otherwise, we continue to play greedy until the game ends or there is nothing left to gain by greedy play. While tit for tat is designed specifically for the prisoner's dilemma game, the characterization has been applied to strategies such as "live-and-let-live", a policy describing the non-aggressive behavior of troops in WWI trenches. [35]

One key point is that this strategy is not trying to match the opponent's existing army stock, K'_f. In tit-for-tat, past events are ignored. Instead, this strategy will match the flow of new K investments, spending the remainder on L or T. If done successfully, the accumulated combat forces should be roughly equal, $K_f \approx K'_f$. At the same time, CUNYbot should exceed the growth rate of its opponent: $L_f > L'_f$. In the case of perfect information, it is easy to infer the investments of an optimizing opponent:

$$\begin{aligned} \alpha'_{K,f} &= \frac{k'}{1+k'} \\ \alpha'_{L,f} &= 1 - \alpha'_{K,f} \end{aligned} \tag{4}$$

Unfortunately, the opponent's investments are frequently obscured by the fog of war, and subject to incomplete information. There has been research evaluating unknown enemy expenditures during early game play, but few approaches evaluate through the entire game. [36], [29]

Typical economic models have historically assumed an exogenous growth rate for labor. I similarly assume an exogenous growth rate for labor in the short run. I assume all known enemy depots (which manufacture workers) are always working on one worker, until they reach a maximum upper limit. This assumption works particularly well on most AIs, and it is standard advice to new RTS players to always build workers. [37] Initially, each player starts with four workers and one depot, while in the late game, the maximum bound in the literature of expert play appears to be under 85 workers. [38] During the game it is a simple matter to subtract destroyed workers (or workers morphed into buildings) and remove them from the total. Thus, the enemy's estimated stock of workers at any time is roughly:

$$\begin{aligned} \hat{w}_0 &= 4 \\ \hat{w}_f &= max(\hat{w}_{f-1} + \frac{enemy\_depots_f}{worker\_build\_time} - destroyed_f, 85) \\ \hat{L_f}' &= \hat{w}_f * value\_of\_a\_worker \end{aligned} \tag{5}$$

On the other hand, the estimated value of the opponent's capital, $\hat{K_f}'$ is assumed to be equal to the stock of the opponent's known combat units, eg. nothing is hidden behind the fog of war. This simplifying assumption is enabled by CUNYbot's unit movement toward the enemy if it is not army-starved. This serves the purpose of scouting. If the units reveal a large $\hat{K_f}'$, CUNYbot will eventually retreat or reinforce its front line, but a better estimate of $\hat{K_f}'$ is gained.

These two modest measures allow the bot to operate in a domain of incomplete information.[11] Reactive CUNYbot then has the information to evaluate $\alpha_{\hat{K},f}{}'$ (and therefore $\alpha'_{L,f}$) every 15 game seconds. It then matches the enemy's investment in aggression by $\alpha_{K,f} \leftarrow \alpha_{K,f} + r(\alpha'_{K,f} - \alpha_{K,f})$. Here, I would expect r be the slowest possible adaptation rate, since that would be greediest, but I do not assume its value. So, r will begin as $r \sim iid.U(0,1)$, and be chosen through the same genetic algorithm already discussed above.[12]

Using this reactive structure, we begin training the starting genomes of CUNYbot again. The final result is visible in Table II. Once again, the bot is reactive, so the values recorded in the genome are the starting conditions.

First and foremost, the reactive CUNYbot evolved to prefer significant reaction rates, moving towards the opponent's value of α'_K at a rate of roughly 30% or more for all 3 built-in AI races. The reactive CUNYbot has a final population of significantly higher α_L and lower α_K in all cases, suggesting greedy play. Furthermore, the ratio of k is lower significantly in nearly all cases.[13] This greedier play is in keeping with CUNYbot's new ability to react to threats by increasing k and building military units. In short, the reactive CUNYbot has learned a tit-for-tat type strategy, in contrast to the aggressive approach learned by the static CUNYbot.

One may also note that the technology level α_T (and the associated t) is lower overall for the reactive CUNYbot. The reduction in α_T was not an anticipated result since α_T was not moved in reaction to enemy decisions. However, CUNYbot's heuristic unit selection prefers expensive units when available, which have long build times. It may be that keeping the technology level low is implicitly allowing CUNYbot to respond quickly via rapidly building units, like Zerglings. The low level of technology, therefore, may be implicitly permitting quicker reactions to threats. Further iterations of CUNYbot will explore this relationship between technology choice and the tit-for-tat style strategy more carefully.

VI. Performance Against Other Bots

In the CIG tournament, CUNYbot will be competing against several other bots who are also arguably superior to the built-in AI. Using SC-Docker [39], CUNYbot has been extensively trained against a large cross section of bots available on SSCAIT. This training should be a useful gauge for the incoming potential contestants in CIG 2018. Starting from a list of all

[11] Future versions may benefit from replay review to extract estimates of worker counts and army sizes despite incomplete information. [29]

[12] In the case CUNYbot cannot evaluate $\alpha'_{K,f}$ (ex. the enemy has no known living units), CUNYbot defaults $\alpha_{K,f}, \alpha_{L,f}$ to the initial values contained in g.

[13] The only race k is not significantly lower is Z. In the previous static CD exploration, Z was by far the lowest k already, and the distribution of k will have relatively heavy tails.

TABLE II
REACTIVE CD LEARNING - FINAL 50 GENOMES WITH 1000 TRAINING ROUNDS

Mean	α_K	α_L	α_T	r	k	t
Reactive CD vs P	0.39	0.61	0.30	0.31	0.64	0.19
W:54.3%	(0.00)	(0.00)	(0.01)	(0.01)	(0.00)	(0.01)
Reactive CD vs T	0.51	0.49	0.45	0.29	1.05	0.47
W:75.8%	(0.01)	(0.01)	(0.01)	(0.01)	(0.03)	(0.01)
Reactive CD vs Z	0.45	0.55	0.40	0.57	0.82	0.33
W:100.0%	(0.00)	(0.00)	(0.00)	(0.00)	(0.01)	(0.01)

Note: SD of each genome element in parentheses. Win rate is evaluated starting from the time the first member of the final generation is added to the end of the experiment.

active bots [40], I excluded bots that had problems with SC-Docker, for a total of 84 potential opponents.[14] SC-Docker comes with a large cross-section of maps, used frequently on SSCAIT.[15] Learning was compartmentalized by opponent, race, and map. That is to say, if an opponent is new, CUNYbot references its history of games on that race and map. If a map is also new, CUNYbot references its history against the race only. CUNYbot played against a randomly selected bot on a randomly selected map. CUNYbot played 2000 such games in serial at a game speed of 1. The results of these training games are listed below in Table III, sorted by win percentage.

The overall win percentage is 28%, though the win rate varies tremendously between opponents. CUNYbot typically beats OpprimoBot, NUSbot and Hannes Bredburg under these conditions with win rates 60%+. Closely matched opponents with win rates between 40-60% include: Lukas Moravec, KaonBot, Sungguk Cha and KillAlll. CUNYbot remains unable to defeat some opponents within the sample window, such as the top-rated Krasi0. A larger portion of CUNYbot's losses are related to David Churchill (UAB) and its descendants (such as Steamhammer or Arrakhammer). However, it has managed some exceptional victories. CUNYbot boasts a substantial 31% win rate against tscmooz, who on SSCAIT has defeated all previous versions of CUNYbot. This result and other victories are cause for optimism, but there is more work to be done on CUNYbot.

VII. CONCLUSION

This paper presents a working summary of CUNYbot, a submission to the 2018 IEEE:CIG StarCraft: Brood War tournament. It describes the operation of CUNYbot in close detail, in particular the functioning of the Cobb-Douglas model underlying its behavior. This model is not specific to StarCraft: Brood War, but has broad application to RTS games in general. I show that the CD model allows the bot to evaluate the game state successfully. I train CUNYbot to play the three built-in AI races with static CD parameters, and the result is three unique sets of relatively aggressive parameters. By contrast, when given the ability to react to the opponent's expenditures, CUNYbot attunes to primarily to greedy parameters. Encouragingly, the greedy style of the reactive CUNYbot matches a tit-for-tat style strategy - it plays friendly when possible, and only builds troops in response to aggression. In each of the six training cases, the CD parameters were significantly different from one another, suggesting important stylistic refinements are being made. Since this exploration has been fruitful, future versions of CUNYbot will expand the use of formal macroeconomic models in the RTS game domain.

[14]The list of bots is visible below in Table III. The removed bots were Bereaver, Sijia Xu, StyxZ, Aurelien Lermant, AyyyLmao, DAIDOES, Zercgberht, DaleeTYC, ggBot.

[15]These maps were the default SSCAIT maps: (2)Benzene.scx, (4)Andromeda.scx, (4)Icarus.scm, (2)Destination.scx, (4)Circuit Breaker.scx, (4)Jade.scx, (2)Heartbreak Ridge.scx, (4)Electric Circuit.scx, (4)La Mancha1.1.scx, (3)Neo Moon Glaive.scx, (4)Empire of the Sun.scm, (4)Python.scx, (3)Tau Cross.scx, (4)Fighting Spirit.scx, (4)Roadrunner.scx

ACKNOWLEDGMENT

The author would like to thank the SSCAIT community, which was indispensable in the creation of this project.

REFERENCES

[1] C. W. Cobb and P. H. Douglas, "A theory of production," *The American Economic Review*, vol. 18, no. 1, pp. 139–165, 1928.
[2] C. I. Jones, "The shape of production functions and the direction of technical change," *The Quarterly Journal of Economics*, vol. 120, no. 2, pp. 517–549, 2005.
[3] D. Acemoglu, "Labor-and capital-augmenting technical change," *Journal of the European Economic Association*, vol. 1, no. 1, pp. 1–37, 2003.
[4] A. R. Tavares, G. L. Zuin, H. Azp, L. Chaimowicz *et al.*, "Combining genetic algorithm and swarm intelligence for task allocation in a real time strategy game," *SBC Journal on Interactive Systems*, vol. 8, no. 1, pp. 4–19, 2017.
[5] D. Christensen, H. O. Hansen, J. P. C. Hernandez, L. Juul-Jensen, K. Kastaniegaard, and Y. Zeng, "A data-driven approach for resource gathering in real-time strategy games," in *International Workshop on Agents and Data Mining Interaction*. Springer, 2011, pp. 304–315.
[6] V. Harmon, "An economic approach to goal-directed reasoning in an rts," *AI Game Programming Wisdom*, vol. 1, pp. 402–410, 2002.
[7] H. Hotelling, "Stability in competition," in *The Collected Economics Articles of Harold Hotelling*. Springer, 1990, pp. 50–63.
[8] H. A. Eiselt and G. Laporte, "Sequential location problems," *European Journal of Operational Research*, vol. 96, no. 2, pp. 217–231, 1997.
[9] A. Rapoport, A. M. Chammah, and C. J. Orwant, *Prisoner's dilemma: A study in conflict and cooperation*. University of Michigan press, 1965, vol. 165.
[10] R. Axelrod and W. D. Hamilton, "The evolution of cooperation," *science*, vol. 211, no. 4489, pp. 1390–1396, 1981.
[11] A. A. Cournot, *Researches into the Mathematical Principles of the Theory of Wealth*. Macmillan, 1897.
[12] H. A. Simon and J. Schaeffer, "The game of chess," *Handbook of game theory with economic applications*, vol. 1, pp. 1–17, 1992.
[13] E. Castronova, "On virtual economies," 2002.
[14] R. Heeks, "Understanding" gold farming" and real-money trading as the intersection of real and virtual economies," *Journal For Virtual Worlds Research*, vol. 2, no. 4, 2009.
[15] E. Castronova, D. Williams, C. Shen, R. Ratan, L. Xiong, Y. Huang, and B. Keegan, "As real as real? macroeconomic behavior in a large-scale virtual world," *New Media & Society*, vol. 11, no. 5, pp. 685–707, 2009.

TABLE III
PLAYING AGAINST SSCAIT BOTS

Bot Name	Win %	Games	Bot Name	Win %	Games	Bot Name	Win %	Games
Bjorn P Mattsson	100	17	tscmooz	31	26	Antiga	5	21
ForceBot	100	26	Ecgberht	25	24	Arrakhammer	5	21
Gaoyuan Chen	100	23	Jakub Trancik	24	21	ICELab	5	21
Marek Kadek	100	28	Florian Richoux	23	13	Microwave	5	21
Tomas Cere	100	28	MadMixZ	23	31	PurpleSpirit	5	21
Goliat	93	29	NiteKatP	22	18	PurpleSwarm	5	20
Hao Pan	91	23	Carsten Nielsen	21	19	tscmoo	5	20
OpprimoBot	90	29	Niels Justesen	19	27	CherryPi	4	28
Fifouille Legend Random	84	32	MadMixP	17	29	Iron bot	4	28
Korean	78	23	Middle School Strats	16	19	McRaveZ	4	27
Stone	76	33	UPStarCraftAI 2016	14	28	Neo Edmund Zerg	4	24
Travis Shelton	76	25	CasiaBot	12	24	Steamhammer	4	26
NUS Bot	75	16	NiteKatT	12	26	tscmoor	4	25
Alice	67	18	GuiBot	11	18	Flash	3	34
Marine Hell	67	27	Matej Istenik	11	36	Zia bot	3	34
WillBot	67	21	skyFORKnet	11	19	Andrew Smith	0	27
Johan Kayser	62	26	Andrey Kurdiumov	10	21	BananaBrain	0	23
Hannes Bredberg	61	23	Roman Danielis	10	20	Black Crow	0	25
KillAlll	53	32	ZurZurZur	10	21	Dave Churchill	0	20
Sungguk Cha	50	28	Sparks	9	23	krasi0	0	24
KaonBot	47	17	Simon Prins	8	26	Locutus	0	24
Lukas Moravec	42	24	Dawid Loranc	7	27	NLPRbot	0	32
Junkbot	39	33	WillyT	7	27	PeregrineBot	0	27
igjbot	38	32	Chris Coxe	6	36	PurpleWave	0	19
MegaBot2017	38	21	Pineapple Cactus	6	31	Tomas Vajda	0	26
Kruecke	33	27	Randomhammer	6	17	TyrProtoss	0	27
MadMixT	33	18	Toothpick Cactus	6	16	WuliBot	0	20
Yuanheng Zhu	33	21						
Total Games: 2000 Overall Win %: 28								

[16] E. Kosminsky, "World of warcraft: The viability of massively multiplayer online role-playing games as platforms for modeling and evaluating perfect competition," *Journal For Virtual Worlds Research*, vol. 2, no. 4, 2009.

[17] Y. F. De Sousa and A. Munro, "Truck, barter and exchange versus the endowment effect: Virtual field experiments in an online game environment," *Journal of Economic Psychology*, vol. 33, no. 3, pp. 482–493, 2012.

[18] M. El-Shagi and G. von Schweinitz, "The diablo 3 economy: an agent based approach," *Computational Economics*, vol. 47, no. 2, pp. 193–217, 2016.

[19] S. Ontanón, G. Synnaeve, A. Uriarte, F. Richoux, D. Churchill, and M. Preuss, "A survey of real-time strategy game ai research and competition in starcraft," *IEEE Transactions on Computational Intelligence and AI in games*, vol. 5, no. 4, pp. 293–311, 2013.

[20] J. Blackford and G. B. Lamont, "The real-time strategy game multi-objective build order problem." in *AIIDE*, 2014.

[21] D. Churchill and M. Certicky, "The current state of starcraft ai competitions and bots," 2017.

[22] P. Garćıa-Sánchez, A. Tonda, A. M. Mora, G. Squillero, and J. Merelo, "Towards automatic starcraft strategy generation using genetic programming," in *Computational Intelligence and Games (CIG), 2015 IEEE Conference on*. IEEE, 2015, pp. 284–291.

[23] D. W. Aha, M. Molineaux, and M. Ponsen, "Learning to win: Case-based plan selection in a real-time strategy game," in *International Conference on Case-Based Reasoning*. Springer, 2005, pp. 5–20.

[24] N. Justesen and S. Risi, "Learning macromanagement in starcraft from replays using deep learning," in *Computational Intelligence and Games (CIG), 2017 IEEE Conference on*. IEEE, 2017, pp. 162–169.

[25] J.-L. Hsieh and C.-T. Sun, "Building a player strategy model by analyzing replays of real-time strategy games," in *Neural Networks, 2008. IJCNN 2008.(IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*. IEEE, 2008, pp. 3106–3111.

[26] J. Kim, K. H. Yoon, T. Yoon, and J.-H. Lee, "Cooperative learning by replay files in real-time strategy game," in *International Conference on Cooperative Design, Visualization and Engineering*. Springer, 2010, pp. 47–51.

[27] H. Köstler and B. Gmeiner, "A multi-objective genetic algorithm for build order optimization in starcraft ii," *KI-Künstliche Intelligenz*, vol. 27, no. 3, pp. 221–233, 2013.

[28] N. Justesen and S. Risi, "Continual online evolutionary planning for in-game build order adaptation in starcraft," in *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, 2017, pp. 187–194.

[29] H. Oh, T. Budianto, Y. Ding, Z. Long, and T. Utsuro, "Identifying the rush strategies in the game logs of the real-time strategy game starcraft-ii," *Training*, p. 306, 2017.

[30] Heinermann, Adam, "Bwapi," 2018. [Online]. Available: https://github.com/bwapi/bwapi

[31] G. Synnaeve and P. Bessiere, "Special tactics: A bayesian approach to tactical decision-making," in *Computational Intelligence and Games (CIG), 2012 IEEE Conference on*. IEEE, 2012, pp. 409–416.

[32] Unknown author, "Chaoslauncher," 2011. [Online]. Available: https://github.com/masterofchaos/chaoslauncher

[33] Z. Michalewicz, G. Nazhiyath, and M. Michalewicz, "A note on usefulness of geometrical crossover for numerical optimization problems." *Evolutionary programming*, vol. 5, no. 1, pp. 305–312, 1996.

[34] Z. Michalewicz and M. Schoenauer, "Evolutionary algorithms for constrained parameter optimization problems," *Evolutionary computation*, vol. 4, no. 1, pp. 1–32, 1996.

[35] P. R. Milgrom, "Axelrod's" the evolution of cooperation"," 1984.

[36] H. Park, H.-C. Cho, K. Lee, and K.-J. Kim, "Prediction of early stage opponents strategy for starcraft ai using scouting and machine learning," in *Proceedings of the Workshop at SIGGRAPH Asia*. ACM, 2012, pp. 7–12.

[37] Attributed to Sean "Day9", Plott , "Liquidpedia," 2018. [Online]. Available: http://liquipedia.net/starcraft2/Mental_Checklist

[38] Attributed to Mike "Bakuryu" Lange, "Liquidpedia," 2018. [Online]. Available: http://liquipedia.net/starcraft/Zerg_Economic_Management

[39] M. Šustr, J. Malý, and M. Čertický, "Multi-platform Version of StarCraft: Brood War in a Docker Container: Technical Report," 2018.

[40] K. Krastev, "Sscait ladder ratings for the last 4000 games," https://cachedsscaitscores.krasi0.com/eloRatings/, 2018.

Forward Model Approximation for General Video Game Learning

Alexander Dockhorn
Otto von Guericke University
Magdeburg, Germany
alexander.dockhorn@ovgu.de

Daan Apeldoorn
Z Quadrat GmbH
Mainz, Germany
daan.apeldoorn@z-quadrat-mainz.de

***Abstract*—This paper proposes a novel learning agent model for a General Video Game Playing agent. Our agent learns an approximation of the forward model from repeatedly playing a game and subsequently adapting its behavior to previously unseen levels. To achieve this, it first learns the game mechanics through machine learning techniques and then extracts rule-based symbolic knowledge on different levels of abstraction. When being confronted with new levels of a game, the agent is able to revise its knowledge by a novel belief revision approach. Using methods such as Monte Carlo Tree Search and Breadth First Search, it searches for the best possible action using simulated game episodes. Those simulations are only possible due to reasoning about future states using the extracted rule-based knowledge from random episodes during the learning phase. The developed agent outperforms previous agents by a large margin, while still being limited in its prediction capabilities. The proposed forward model approximation opens a new class of solutions in the context of General Video Game Playing, which do not try to learn a value function, but try to increase their accuracy in modelling the game.**

***Index Terms*—Forward Model Approximation, General Video Games, exception-tolerant Hierarchical Knowledge Bases, Belief Revision, Monte Carlo Tree Search, Breadth First Search**

I. Introduction

The development of General Intelligence (i. e., intelligent algorithms that are able to cope with multiple kinds of different problems) is one of the key long term goals in Artificial Intelligence (AI) research. To evaluate approaches for this long term goal, in recent years, the General Game Playing challenge [1] and (more recently) the General Video Game Playing Artificial Intelligence (GVGAI) competition [2] have been developed, where agents have to play several different types of games which are not (concretely) known in advance.

In this paper, we present a novel approach of a learning agent which learns to play different video games in the context of the GVGAI competition [2]. To achieve this, we combine multiple concepts from both the symbolic and the subsymbolic artificial intelligence communities using machine learning together with belief revision and action selection. More precisely, our agent first uses statistical methods to learn the game mechanics in form of an *approximated forward model*. In this study we use *exception-tolerant Hierarchical Knowledge Bases* (HKBs) [3], containing rules on multiple levels of abstraction, to learn and store the approximated forward model. These knowledge bases are later revised when the agent is confronted with new or changed environments (e. g., new levels) by a novel belief revision approach for HKBs.

Those knowledge bases can be used while playing the game for reasoning about the value of future states. Based on previous experiences we can partially simulate the outcome of our future actions, enabling us to use well-known search schemes for action selection. In this study we use Monte Carlo Tree Search (MCTS) and Breadth First Search (BFS) to evaluate future states. Finally, we choose the action with the highest expected outcome and return it for execution.

The main contributions of the paper are:

- a learning approach which is able to quickly learn the mechanics of dynamic environments (e. g., games) in form of an approximated forward model
- representing and storing the approximated forward model in human readable format as HKBs (i. e., rule-based knowledge which is organized on multiple levels of abstraction)
- a simple belief revision approach for HKBs
- an agent architecture using the generated approximated forward model with a flexible action selection mechanism
- an agent model to be used for the GVGAI learning-track

Section II briefly outlines related work and in Section III the preliminaries needed for our agent model will be introduced. After that, in Section IV, we describe the details of creating and using an approximated forward model in context of an agent for the GVGAI competition. In Section V, we evaluate and compare the proposed approach with other known agents based on several games from the GVGAI competition. A conclusion and an outlook on future work are given in Section VI.

II. General Game Playing

Next to the many efforts put into artificial intelligence methods for various board games such as Chess, Morris, and Go, the Stanford General Game Playing competition [1] asked its participants to implement agents, which can compete in a set of previously unknown games. Playing agents needed to act based on the current state of the board and a set of simple rules. Many attempts have been made to create similar description languages for the definition of video games.

Thanks to the efforts of Tom Schaul, the Video Game Definition Language [4] was created. Based on his work the General Video Game AI (GVGAI) competition was created. It offers a framework, in which many arcade like video games

978-1-5386-4359-4/18/$31.00 ©2018 IEEE

were replicated. Here, an agent receives a state description and a set of up to five possible actions. Those actions are named after buttons of a typical game controller and map the buttons ”Up”, ”Down”, ”Right”, ”Left”, ”Action”. However, the outcome of each action is completely dependent on the rules of the game, such that a suitable policy needs to be learned for each problem individually.

In recent years the competition offered multiple tracks, which focus on different aspects of general video game playing. The next sections quickly summarize the single player game playing and the learning track. In the remainder of this paper we will focus on the study of the learning track.

A. GVGAI - Single Player Game Playing Track

The currently most popular track is the single player game playing track. Here, a forward model is provided to the agents, which can be used to simulate the outcome of an action.

In 2017 a total of 22 submissions entered the competition (including 5 sample submission provided by the competition hosts). The currently best performing agent, Yolobot [5], uses a mix of BFS and MCTS. While the former is used in games, which include deterministic state transitions, the latter is applied to all non-deterministic games. Additionally, the algorithm identifies reachable objects and rates its interest on them. Either BFS or MCTS is used to find a suitable path to potential targets, while avoiding any dangers.

Due to the inclusion of non-deterministic games, the application of MCTS variants such as Open Loop MCTS (OLMCTS) were studied in multiple works (see a summary of agents in [6]). OLMCTS and other tree searching algorithms such as Open Loop Expectimax Tree Search are able to quickly sample possible action sequences and evaluate their outcome [5].

Next to tree search algorithms, the search of action sequences based on genetic algorithms showed to be a popular choice. Methods such as the Rolling Horizon Algorithm [7] evolve short action sequences and evaluate their outcome based on a scoring function similar to OLMCTS.

The overall performance of agents in the game playing track is already very good. Despite being confronted with a previously unknown game, the agents are often able to win a game or at least find action sequences, which yield a high score.

B. GVGAI - Single Player Game Learning Track

In contrast to the game playing track, the game learning track has an increased difficulty due to the forward model being removed as a source of information from the agent. All previously discussed methods heavily rely on such a forward model, which enables the agent to run simulations for all possible actions. Therefore, new algorithms need to be found for solving problems of this domain.

The agent developed by Ercüment İlhan [8] uses an MCTS agent, which was enhanced with an online on-policy temporal-difference learning method, called true online Sarsa(λ) [9]. Here, the agent optimizes its estimation of the state-action value function by continuously revising it based on the repeated interactions with the game environment. This agent performed best in the training set, but only placed 5th in the evaluation game set (slightly ahead of an adaptation of the Yolobot from the game playing track). Not much information is available on the other agents.

Surprisingly the random controller provided by the competition organizers performed second best in the evaluation set, just three points worse than the first place. This shows that well-known methods for value estimation or policy improvement do not work at their full potential using the limited information in this track. Short time frames for learning the games rules ($\leq$ 5 minutes) and decision-making (40ms) drastically limit the applicability of iterative methods or long search procedures. Hence, there is still more room for improvement.

Our work is motivated by the huge performance gap between agents of the game playing and game learning track and the idea that machine learning combined with methods from symbolic knowledge representation can contribute to create an appropriate agent model. This work will show how the introduction of *approximated forward models* and efficient learning and operation schemes can help us in developing new kinds of agents.

III. Preliminaries

This section will briefly outline the preliminaries and components needed for our agent model introduced in Section IV for the GVGAI learning track. We will first explain the basics needed for our novel *approximated forward model* for which it is useful to define Hierarchicle Knowledge Bases (HKB) and how they can be learned (Subsections III-A to III-C). These sections will closely follow several preliminary works, especially [3], [10], [11]. After that, the reasoning algorithm defined on HKBs will be briefly summarized, following [11] (Subsection III-D). Subsequently, we will introduce some *modifications* on HKBs and the learning process (Subsection III-E) to fit the needs of our agent model. Finally, we will propose a belief revision approach for HKBs (Section III-F) and describe a module for action selection (Subsection III-G).

A. Basic Agent Model for HKBs

We consider an agent which is equipped with n sensors through which it can perceive its current state in the environment (e. g., a game) and which is able to perform actions from a predefined action space (e. g., the keys to be pressed on the controller). Furthermore, the agent can perceive whether the performed actions were good, in form of (numeric) rewards. The perceived rewards can then be used to learn a weighted state-action pair representation where the highest weight determines which action has to be performed, given a perceived state.

More formally, in such a representation, a state s is an element of a multi-dimensional state space $\mathbb{S} = \mathbb{S}_1 \times ... \times \mathbb{S}_n$ where n is the number of the agent's sensors and every $\mathbb{S}_i$ is a set of possible values of the corresponding sensor. Furthermore, the agent selects actions from a predefined action set $\mathbb{A}$ and the learned weights are stored in a multi-dimensional matrix

$\hat{Q} = (q_{s_1,...,s_n,a})$ with $s_i \in \mathbb{S}_i$ and $a \in \mathbb{A}$. The weights can be learned by different machine learning approaches, provided that the learning approach results in a representation such that given a state, the highest weight determines the best action to be selected (i. e., $a^{\max}_{s_1,...,s_n} = \arg\max_{a' \in \mathbb{A}} q_{s_1,...,s_n,a'}$).

After selecting an action the environment responds with a reward and a successor state s_{t+1}. In general a game can be modelled as a probability distribution over $P(s_{t+1}, r_{t+1}|s_0, a_0, r_1, s_1, a_1, \ldots, r_t, s_t, a_t)$, which maps the probability of each successor state and its accompanied reward depending on all previous interactions. Analysing or storing such a probability distribution is near to impossible, due to its exponentially growing complexity. When the process to be analysed fulfils the Markov Property, the probability distribution reduces to $P(s_{t+1}, r_{t+1}|s_t, a_t)$, in which each successor state and its reward is only dependent on the environment's last state and the agent's last chosen action. Our target will be to construct a classifier, hereinafter referred to as *approximated forward model*, which approximates the distribution $P(s_{t+1}, r_{t+1}|s_t, a_t)$ based on previous interactions with the environment.

Such an *approximated forward model* can be split into multiple sub-components in case the multi-dimensional representation $\mathbb{S}$ consists of independent components. If this is the case a complete reconstruction of the state $\mathbb{S}$ can be achieved by modelling each independent component separately.

The following subsections introduce HKBs, which will be used for learning and revising such classifiers during continuous interactions with the games environment.

B. Definition of HKBs

In [10], an extraction approach is proposed which is able to extract an HKB from a weighted state-action pair representation (where the maximum weight determines the best action, given a state). This section only provides the main definitions needed to understand the basic idea of HKBs and how they can be learned from weighted state-action pairs by closely following [3], [10], [11]. For more details on HKBs, the reader should refer to the original literature mentioned here. An HKB consists of rules which are organized on different levels of abstraction.

An HKB can handle multiple rules per level and the rules also comprise weights (in contrast to *Exception Lists* [12]). According to [11] two different kinds of states and two different kinds of rules need to be distinguished:

Definition 1 (Complete States/Partial States) *A complete state is a conjunction $s := s_1 \wedge ... \wedge s_n$ of all values s_i currently perceived by an agent's sensors, where n is the number of sensors and every perceived sensor value $s_i \in \mathbb{S}_i$ of the corresponding sensor value set $\mathbb{S}_i$ is assumed to be a fact in the agent's current state. A partial state is a conjunction $s := \bigwedge_{s' \in S} s'$ of a subset $S \subset \{s_1, ..., s_n\}$ of the sensor values of a complete state.*

Definition 2 (Complete Rules/Generalized Rules) *Complete rules and generalized rules are of the form $p_\rho \Rightarrow a_\rho\ [w_\rho]$, where p_ρ is either a complete state (in case of an complete rule) or a partial state (in case of a generalized rule). The conclusion $a_\rho \in \mathbb{A}$ is an action of the agent's action space $\mathbb{A}$ and $w_\rho \in [0, 1]$ is the rule's weight.*

Thus, *complete rules* map *complete states* to actions and *generalized rules* map *partial states* to actions. An HKB can now be defined as follows:

Definition 3 (Exception-Tolerant Hierarchical Knowledge Base) *An exception-tolerant Hierarchical Knowledge Base (HKB) is an ordered set $\mathcal{KB} := \{R_1, ..., R_{n+1}\}$ of $n+1$ rule sets, where n is the number of sensors (i. e., the number of state space dimensions). Every set $R_{i<n+1}$ contains generalized rules and the set R_{n+1} contains complete rules, such that every premise $p_\rho = \bigwedge_{s \in S_\rho} s$ of a rule $\rho \in R_i$ of length $|S_\rho| = i - 1$.*

According to Definition 3, the set R_1 contains the most general rules (with empty premises) and the set R_{n+1} contains the most specific (i. e., complete) rules.

For the relations of rules, the term of *needed exception* will be used, according to the following definition (cf. [3]):

Definition 4 (Needed Exception) *A rule $\rho \in R_{j>1}$ is an exception to a rule $\tau \in R_{j-1}$ with premise $p_\tau = \bigwedge_{s \in S_\tau} s$, action a_τ as conclusion and weight w_τ, if $S_\tau \subset S_\rho$ and $a_\rho \neq a_\tau$. The exception is needed, if no other rule $\upsilon \in R_{j-1}$ exists with premise $p_\upsilon = \bigwedge_{s \in S_\upsilon} s$ and action a_υ as conclusion where $S_\upsilon \subset S_\rho$, $a_\upsilon = a_\rho$ and $w_\upsilon > w_\tau$.*

C. Learning HKBs

An HKB can be extracted from a weighted state-action pair representation $\hat{Q}$ (that is learned, e. g., through a Reinforcement Learning technique or by simply counting relative frequencies) using the following approach originally introduced in [10], closely following [11] here:

The approach takes a weighted state-action pair representation $\hat{Q}$ as input and returns an HKB $\mathcal{KB}^{\hat{Q}}$ which reflects the knowledge contained in $\hat{Q}$ by performing the following steps:

1) *Initial creation of rule sets:*
   In the first step, the multiple abstraction levels $R_1, ..., R_{n+1}$ of the knowledge base are initially filled with rules. The weights of generalized rules are created by averaging the weights in $\hat{Q}$ over the missing dimensions.[1]
2) *Removal of worse rules:*
   In all sets R_j, a rule $\rho \in R_j$ is removed, if there exists another rule $\sigma \in R_j$ with the same partial state as premise having a higher weight.
3) *Removal of worse more specific rules:*
   In all sets $R_{j>1}$, a rule $\rho \in R_j$ with premise $p_\rho = \bigwedge_{s \in S_\rho} s$, conclusion a_ρ and weight w_ρ is removed, if there exists a more general rule $\sigma \in R_{j'<j}$ with premise $p_\sigma = \bigwedge_{s \in S_\sigma} s$ where $S_\sigma \subset S_\rho = \{s_1, ..., s_{j-1}\}$ and weight $w_\sigma \geq w_\rho$.
4) *Removal of too specific rules:*
   In all sets R_j, a rule $\rho \in R_{j>1}$ with premise $p_\rho = \bigwedge_{s \in S_\rho}$ and conclusion a_ρ is removed, if there exists a more

[1] For performance reasons, only state-action pairs can be considered here that contribute to the best policy found by the preceding learning process (see [3] for a first attempt to a more efficient algorithm that preselects potentially relevant rules in this step).

general rule $\sigma \in R_{j'<j}$ with the same action $a_\sigma = a_\rho$ as conclusion and with premise $p_\sigma = \bigwedge_{s \in S_\sigma} s$ where $S_\sigma \subset S_\rho = \{s_1, ..., s_{j-1}\}$ and if ρ is not a *needed exception* to a rule $\tau \in R_{j-1}$.

5) *Optional filter step:*
Optionally, filters may be applied to filter out further rules which are, e. g., helpful to explain the knowledge contained in $\hat{Q}$ through the optimal found policy so far, but which are not needed for reasoning later.

After performing these steps on $\hat{Q}$, the knowledge base $\mathcal{KB}^{\hat{Q}}$ comprises all sets $R_j \neq \emptyset$ with the extracted rules representing the implicit knowledge contained in the learned weights of $\hat{Q}$ in a compact way.

D. Reasoning on HKBs

This section briefly summarizes the basic idea of the efficient reasoning algorithm on HKBs which was first introduced in [10]. The summary closely follows [11]:

Given perceived sensor values $s_1, ..., s_n$, the reasoning algorithm searches an HKB upwards (starting from the bottom-most level R_{n+1}) for the first rule, which premise is fulfilled. This rule is then returned as concluding action (see [10] for details). By this, the algorithm selects the most specific rule that fits to the perceived sensor values and falls back to the next more unspecific rule (which serves as a heuristic), in case no more specific rule with a fitting premise could be found.

E. Modifications for Our Agent Model

In this section, the original HKB approach according to [3], [10], [11] (which was outlined in the previous Sections III-A to III-C) will be modified to fit the needs of our agent model for the learning track of the GVGAI competition. A knowledge representation based on exceptions which are layered on several levels of abstraction is a rather useful approach to gain a compact representation of the knowledge about an environment like a game (which can also be exploited during a learning process as has been demonstrated in [10], [11]). Nevertheless, according to the GVGAI competition specification, our learning agent is supposed to work in *multiple* levels of a game and the agent furthermore only sees three out of five levels in the training phase. Thus, the agent should be able to learn the *general mechanics* of the game rather than optimizing its behavior for a single level.

For this purpose, we modify the definitions of HKBs (especially Definition 2) such that rules no longer represent a mapping of a state to an action but a mapping of a state and an action which was performed in that state to a resulting subsequent state.

More formally, rules contained in the HKB are now defined as follows:

Definition 5 (Modified Complete Rules/Generalized Rules) *The modified complete rules and generalized rules are of the form $p_\rho \wedge a \Rightarrow p'_\rho\ [w_\rho]$, where p_ρ is either a complete state (in case of a complete rule) or a partial state (in case of a generalized rule), $a_\rho \in \mathbb{A}$ is an action of the agent's action space $\mathbb{A}$, p'_ρ represents one (or more) sensor value(s) of a subsequent state (resulting from action a performed in state p_ρ) and $w_\rho \in [0, 1]$ is the rule's weight. The sensor values of p'_ρ do not necessarily need to be of the same sensors used for p_ρ.*

Note that since the creation of HKBs from data is computationally rather expensive (cf. [10], see [3] for a first attempt towards a faster algorithm for HKB extraction). In case of our agent model, we will only consider small subsets $\mathbb{S}' \subset \{\mathbb{S}_1, ..., \mathbb{S}_n\}$ of the agent's state space dimensions for p_ρ and p'_ρ in Definition 5. This results in several smaller HKBs where every HKB represents a certain aspect of the agent's collected knowledge about the environment. Furthermore, a merging technique will be used to gain an HKB for higher dimensional state spaces by merging multiple smaller HKBs. (For details see Section IV.)

In addition, the extraction algorithm described in Section III-C will be extended by the following filter at the end of Step 5: All rules $\rho \in R_{j>1}$, which premise does not contain an action will be removed.

In the following, the idea of such modified HKBs will initially be explained in the context of the game *Butterflies* from the GVGAI competition framework [2].

Example 1 (Butterflies) *We consider the game Butterflies from the GVGAI competition framework [2], where an agent has to collect butterflies by touching them (see left part of Figure 1): Every time when collecting a butterfly, the agent's current score is increased by 2. To learn knowledge about the scoring of the game, the agent's surrounding objects and its orientation are considered as a subset of the state space dimensions. Furthermore, the agent's action space is given as usual (i. e., $\mathbb{A} := \{\text{Up}, \text{Down}, \text{Left}, \text{Right}, \text{Use}\,(, \text{None})\}$). After a short training phase, the learned HKB regarding the knowledge of the scoring of the game (i. e., which actions lead to which changes regarding the agent's current score, given a state) is shown in the right part of Figure 1.*

$\mathcal{KB}_1$:

score±0 [0.944]	R_1
above5 ∧ Up → score+2 [1.0] below5 ∧ Down → score+2 [1.0] left5 ∧ Left → score+2 [1.0] right5 ∧ Right → score+2 [1.0]	R_3

Fig. 1. Excerpt from the First Level of the GVGAI Game *Butterflies* with Corresponding HKB of the Agent's Knowledge About the Scoring in the Game After a Short Training Phase

The HKB resulting from Example 1 (right part of Figure 1) can be read as follows: According to the single rule score±0 on the most general level R_1 (which has an empty premise), the agent learned that *in general* (when no action is performed), no score changes are expected. This covers most of the cases as indicated by the high weight. According to the four rules on level R_3, the agent learned that if an object with id 5 (i. e., a butterfly) is perceived above, below, to the left or to the right of

the agent and the agent performs an action in the corresponding direction of the object, then the score is increased by 2. The weights of 1.0 indicate that this should happen in all cases when perceiving these objects and performing these actions. Level R_2 does not contain any rules, thus there are no relevant exceptions from the most general rule on level R_1 in this game that are only based on actions (without considering any surrounding objects). Furthermore, level R_4 is also empty, since there are no relevant exceptions from the rules on level R_3 that additionally involve the orientation of the agent.

F. Belief Revision on HKBs

Unlike *relearning* knowledge that is once gained by subsymbolic machine learning approaches (e. g., during the training phase of a game), a much more efficient solution could be a symbolic revision approach used on a previously learned knowledge base. By this, the learned knowledge can be expanded or changed immediately on a symbolic level instead of relearning statistically on a subsymbolic level. For this purpose, this section introduces a novel belief revision approach for HKBs: The proposed algorithm is a first simple attempt to realize revision on HKBs and leaves a lot of room for improvements. Nevertheless, it allows the agent to revise the learned knowledge represented by an HKB, in case the environment changes.

We consider a learned HKB with the modifications described in Section III-E and the reasoning algorithm described in Section III-D: Given a state representation s and an action a, and given that the representation of a subsequent state s' inferred through the HKB is *inconsistent* with the corresponding representation of the actual perceived subsequent state s'_{per} of the agent (i. e., $s' \not\equiv s'_{\text{per}}$), the basic processing of the revision algorithm is as follows:

- *Adding a new exception:*
  If the rule leading to the inconsistent inference is located on a level $R_{j<n+1}$, then a new rule is added on level R_{n+1} with the updated conclusion according to the actual perceived subsequent state.
- *Exchanging an existing exception:*
  Otherwise, if the inconsistent inference is located on the most specific level R_{n+1}, then the conclusion of that rule is simply replaced by the correct conclusion according to the actual perceived subsequent state.

By this, the learned knowledge about the game mechanics can be quickly adapted to changes in the environment (e. g., scoring distributions, object localizations or even new kinds of objects) in case the agent is being confronted with new levels.

G. Action Selection

After a short learning phase we use the extracted HKBs as approximated forward model to predict the future states after picking action a. The action selection process will be guided by two sub-systems. MCTS is used for a broad exploration of longer action-sequences. In case no preferable solution can be found, we use BFS to find the shortest path to a state, which would yield an increase in points. In the following we will shortly review MCTS. The combination of both sub-systems and the adaptations made will be introduced in Section IV.

1) Monte Carlo Tree Search (MCTS): In this study we will make use of MCTS for sampling long-term action sequences. MCTS is a heuristic search algorithm, which consists of four phases, (1) node selection, (2) node expansion, (3) simulation, and (4) backpropagation of the (expected) reward. The first two steps form the tree policy, while the latter two are also known as the default policy. The simulation during phase (3) consists of multiple rollouts of action sequences, which are simulated using a forward model. A rollout starts at the agent's current state and repeatedly chooses actions till either the end of an episode, at which the winner of the game is known, or any mid-game state is reached. In case the simulation is stopped before the end of an episode a scoring function is used to evaluate the value of the final state. The result of an action is estimated using a forward model, which describes the transition from state s to the next state s' after using action a. The observed score at the end of an episode is backpropagated to iteratively improve the value estimation of intermediate states. After all simulations are completed the agent uses its value estimate to choose and execute the action with the highest expected return.

Multiple factors influence the capabilities of this search strategy. Next to the quanity and depth of a rollout, previous studies on card games showed that the quality of a rollout is a critical factor for a strong playing behavior [13], [14]. This introduces a trade-off between the depth, quantity and quality of a rollout during the simulation phase. All three will be limited during this study due to the short time span for action selection and the computational overhead of the approximated forward model. Subsection IV-B will discuss our optimizations for MCTS, which facilitate a higher quantity of rollouts.

IV. Agent model

The agent model builds on the preliminaries described in Section III. First, the basic composition of the concepts required to learn the game mechanics will be summarized (Section IV-A). Consequently, this section focuses on how the action selection is realized based on these ideas and which modifications where necessary to optimize the process (IV-B). An overview of the agent model is provided in Figure 2.

A. Basics

Since the learning track of the GVGAI competition is divided into a *training phase* (on three out of five levels of each game) and an *evaluation phase* (on already known and two additional levels), the basic idea is to use the training phase to accumulate knowledge about the game mechanics in form of modified HKBs (as described in Section III-E) and to use planning based on the gained knowledge in the evaluation phase. Furthermore, a belief revision approach is used during the evaluation phase in case new experiences are contradicting the learned knowledge of the training phase in some aspects.

1) Dividing the Learned Knowledge into Different Aspects: Games can vary widely in their game play and the goals that have to be reached to win the game. Thus, different aspects of

the game mechanics are important depending on the kind of game that is played. For this purpose, the knowledge about the game mechanics will be stored in three different HKBs – one HKB for one type of knowledge representing one aspect that might be relevant for decision-making. The following three aspects are covered:

- $\mathcal{HKB}_{\text{move}}$*, Relative Movement*: The movement depending on the relative position to other objects (e. g., obstacles like "objects of this type cannot be passed").
- $\mathcal{HKB}_{\text{scr}}$*, Scoring*: Score changes depending on interactions with other objects (e. g., beneficial objects like "collecting objects of this type increases the score by X").
- $\mathcal{HKB}_{\text{win}}$*, Winning/Losing*: Which kind of object interactions lead to winning or losing the game (e. g., objects that lead to winning the game when touching them).

The division of the forward model into multiple HKBs can be justified in case the game's individual components are independent from another. In this case the complete model is reconstructible from the outputs of each sub-model.

2) Fast Creation of HKBs for the Different Aspects: As mentioned in Section III-C and Section III-E, creating HKBs can be computationally expensive on higher dimensional state spaces. To overcome this problem, for every HKB contained in the meta knowledge base, multiple separate HKBs are created from reduced state spaces with less dimensions. The resulting HKBs are than merged to one final HKB representing one of the three types of knowledge $\{\mathcal{HKB}_{\text{move}}, \mathcal{HKB}_{\text{scr}}, \mathcal{HKB}_{\text{win}}\}$.

In the following, the creation of the *Scoring* HKB $\mathcal{HKB}_{\text{scr}}$ will be exemplarily described (the process is very similar for the other two types of knowledge in the meta knowledge base):

$\mathcal{HKB}_{\text{scr}}$ reflects the knowledge about which action leads to which score change, given the orientation of the agent and the types of the objects currently surrounding it. The HKB $\mathcal{HKB}_{\text{scr}}$ could be created (according to the algorithm described in Section III-C) from the matrix $\hat{Q}_{\text{scr}} = (q_{s_{\text{above}}, s_{\text{below}}, s_{\text{left}}, s_{\text{right}}, s_{\text{ori}}, a, s_{\text{scr}}})$ with $s_{\text{above}}, s_{\text{below}}, s_{\text{left}}, s_{\text{right}} \in \mathbb{S}_{\text{obj}}$, $s_{\text{ori}} \in \mathbb{S}_{\text{ori}}$, $a \in \mathbb{A}$ and $s_{\text{scr}} \in \mathbb{S}_{\text{scr}}$, where $\mathbb{S}_{\text{obj}}$ is the set of object types, $\mathbb{S}_{\text{ori}}$ is the set of the agent's orientations, $\mathbb{A}$ is the agent's action space and $\mathbb{S}_{\text{scr}}$ is the set of score changes. Every element of $\hat{Q}_{\text{scr}}$ represents a learned relative frequency of how often an action leads to a certain score change, given the agent's orientation and the types of objects *above*, *below*, *left*, and *right* of the agent.

However, instead of creating $\mathcal{HKB}_{\text{scr}}$ directly from the seven-dimensional matrix $\hat{Q}_{\text{scr}}$, as a first step, the four smaller HKBs $\mathcal{HKB}_{\text{scr}}^{\text{above}}$, $\mathcal{HKB}_{\text{scr}}^{\text{below}}$, $\mathcal{HKB}_{\text{scr}}^{\text{left}}$ and $\mathcal{HKB}_{\text{scr}}^{\text{right}}$ are created (each according to the algorithm described in Section III-C). Each of these HKBs represents the learned knowledge about the score change, given the orientation of the agent and a surrounding object focussing only on *one* of the objects currently above, below, left, or right of the agent. Every of the four smaller HKBs is created from an only four-dimensional matrix which contains the learned relative frequencies how often an action leads to a certain score change, given the agent's orientation and the type of one of the objects above, below, left or right of the agent, respectively. In case of $\mathcal{HKB}_{\text{scr}}^{\text{above}}$, this matrix has the form $\hat{Q}_{\text{scr}}^{\text{above}} = (q_{s_{\text{above}}, s_{\text{ori}}, a, s_{\text{scr}}})$ with $s_{\text{above}} \in \mathbb{S}_{\text{obj}}$, $s_{\text{ori}} \in \mathbb{S}_{\text{ori}}$, $a \in \mathbb{A}$ and $s_{\text{scr}} \in \mathbb{S}_{\text{scr}}$, where $\mathbb{S}_{\text{obj}}$ is the set of object types, $\mathbb{S}_{\text{ori}}$ is the set of the agent's orientations, $\mathbb{A}$ is the agent's action space and $\mathbb{S}_{\text{scr}}$ is the set of score changes.

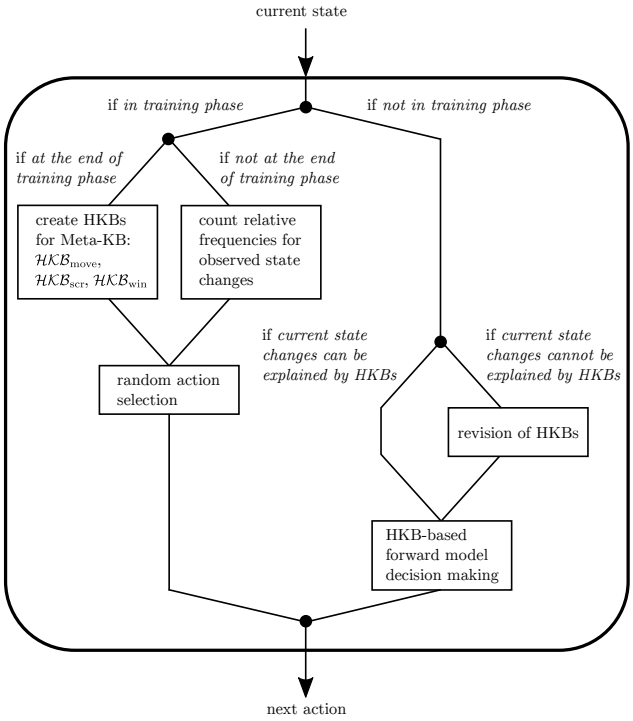


Fig. 2. Overview of the Agent Model Incorporating Learning, Revision, and Action Selection

As the second step, the four smaller HKBs that have been created before are *merged* to the final HKB $\mathcal{HKB}_{\text{scr}}$ by adding all rules of one level to the respective level in the merged HKB $\mathcal{HKB}_{\text{scr}}$. If a rule with the same premise and the same conclusion already exists on this level, then the rule with the smallest weight is kept.

By dividing every HKB in several smaller HKBs (each representing only a subset of the state dimensions of the corresponding complete HKB), the computational challenge of creating the entire meta knowledge base could be overcome and all HKBs of the meta knowledge base can be created in acceptable time.

B. Action Selection

After the training time expired we use and revise the extracted HKBs during the action selection phase. Many agents in the game playing track already based their action selection on MCTS. Applying MCTS in the learning track is not straightforward, since the necessary simulation for the rollouts is not available. We use the generated HKBs to create an approximation of the forward model. The additional time spent on the calculation of future states limits the search depth considerably. During our tests we aimed for 20 simulations using a search depth of 20 actions. Using this approximation we

are able to partially simulate future states. We use a discounted return based on all received simulated rewards between start and end of the rollout. The action, which yields the highest average return is chosen as the agent's next action.

Similar to the agent Yolobot we tested BFS as an alternative to MCTS. However, some adaptations to vanilla breadth first search were necessary to compensate for the additional time spent on calculating the future states and the high number of states to be processed in deeper layers. Since those methods showed to be beneficial in the application of both search procedures, we also added them to the MCTS implementation.

1) Fast Forward Prediction: Many games consist of continuous movements in which the agent needs to use the same action multiple times in a row. We make use of this fact by multiplying the outcome of a relative movement times the block size of the game. This considerably reduces the simulation steps needed for long movements and allows the agent to explore far positions during a single rollout.

2) State Pruning: In general, we cannot be sure which states to prune, since state transitions are only partially simulated. As an unpruned tree grows exponentially in size, it is near to impossible to reach higher search depths using breadth first search in the available time. Therefore, we consider states to be equal in case they yield the same agent position, score, and winner. Using this pruning strategy movements such as "first up, then left" and "first left, then up" are considered to yield the same result and are only processed once.

V. Evaluation

Our approach is evaluated and compared with other agents using multiple games from the GVGAI framework. In Subsection II-B we already summarized the previous agents' frameworks in learning and playing unknown games. Because the implementations of other agents are currently not public, we use the same set of games used in the CIG 2017 training set, for which the competition web page[2] offers detailed results on the agents' performances. For each of the 10 games we train our agent for about 50 seconds cycling through the first 3 levels, in contrast to a maximum of 5 minutes of training time provided by the competition framework. After the first 50 seconds we use the collected observation data to construct our knowledge bases, which takes less than 1 minute in which the agent continuous using random actions. Using the approximated forward model we play 10 rounds on each of the two remaining levels per game. The average score after playing those 20 rounds is reported in Table I. Next to the results of our agent, we list the performances of all the agents of the 2017 GVGAI learning-track competition.

This evaluation is limited due to the unavailability of agent implementations and the limited number of recorded results on the competition web page. Nevertheless, the 10 chosen games of the training set represent a well mixed set of games, which require the agents to play well in a variety of scenarios.

Our approach is the single best agent in 5 out of 10 games and second best in 1 game. We beat the previously best agent by a large degree in games such as *Boulderdash*, *Butterflies*, and *Survive Zombies*. In those games the agent wins by moving on the same or neighbouring position as other objects or characters. Here, the applied search scheme can work to its full potential, because the extracted knowledge predicts future states with high accuracy.

The proposed framework excels in games such as *Butterflies* and *Survive Zombies*. After the agent learns how to move (represented in $\mathcal{HKB}_{\text{move}}$), he easily scores points by searching a path to the nearest objective, which is either a butterfly or a zombie respectively. In the game *Survive Zombies* he also needs to apply knowledge about the winning and losing condition ($\mathcal{HKB}_{\text{win}}$). In case the player runs out of health he quickly needs to collect a healing item while avoiding zombies. As it was the case for chasing, the same search schemes are efficient in avoiding the dangers while searching for an item.

In two of the remaining games (*Frogs*, *Portals*) our agent and all other agents score zero points. This is due to the sparse reward given in both games. For this reason, our agent is not able to rate an action based on the return, because the score does not change till the endpoint of our simulated episode. Hence, we are not able to differentiate between good and bad actions, such that our agent falls back to random behaviour.

The same effect can be observed in the game *Sokoban*, which additionally to the sparse reward requires planning of a long action sequence. Currently learned knowledge bases do not consider movement of non-player entities, which are necessary to solve the puzzles provided in each level. For this reason it is impossible to plan the necessary action sequences to win the game. Hence, the agent once again falls back to choosing actions at random.

Despite the good performance, our agent has still much more room for improvement in games such as *Aliens* and *Boulderdash*. The current score could be improved by taking the actions of non-player characters into account. For example, in the game *Alien* the movement of the aliens and the fired shots can be predicted very easily. Using this information should allow to plan how an enemy can be hit.

To get an overview on the overall performance of our agent we applied the Formula 1 Scoring system, which was already used in the original competition. Here, the agents are ranked based on their average points per game. Depending on their ranking the agents receive 25, 18, 15, 12, 10, 8, or 6 points, whereas the best agent receives the most points. Table II shows the score each agent received per game and its sum over all games. Our results show that the proposed agent outperforms previous agents by a large margin of 32 points.

VI. Conclusion and Future Work

With the help of the proposed *Forward Model Approximation*, we are able to outperform other agents of the GVGAI learning track using only about a fifth of the available learning time, plus less than one additional minute for the creation of the knowledge bases. Our proposed algorithm learns a prediction of future states based on a given state and an action to be applied. This prediction model is split into multiple individual

[2]http://gvgai.net/gvg_rankings_learning_1p.php

TABLE I
AGENT PERFORMANCES ON THE GVGAI-LEARNING TRACK GAMES,
G1 = ALIENS, G2 = BOULDERDASH, G3 = BUTTERFLIES, G4 = CHASE, G5 = FROGS,
G6 = MISSILE COMMAND, G7 = PORTALS, G8 = SOKOBAN, G9 = SURVIVE ZOMBIES, G10 = ZELDA
AGENTS: FMA = FORWARD MODEL APPROXIMATION, DUU = DONTUNDERESTIMATEUCHIHA

Agent Name	Win Rate / Average Points per Game									
	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10
FMA	0.30 / 35.4	**0.00 / 2.1**	**1.00 / 18.7**	**0.00 / 1.1**	**0.00 / 0.0**	0.50 / -1.8	**0.00 / 0.0**	0.10 / 0.9	**0.00 / 1.3**	**0.00 / 0.4**
ercumentilhan	0.45 / 37.3	0.00 / 1.3	0.90 / 18.6	0.00 / 0.7	**0.00 / 0.0**	0.50 / -0.1	**0.00 / 0.0**	0.00 / 0.7	0.00 / 0.1	0.00 / -0.1
sampleRandom	0.10 / 29.8	0.00 / 1.4	0.85 / 19.3	0.00 / 0.5	**0.00 / 0.0**	0.50 / -0.5	**0.00 / 0.0**	0.10 / 0.8	0.00 / -0.1	0.00 / 0.3
sampleLearner	0.20 / 34.5	0.00 / 1.3	0.40 / 15.3	0.00 / 0.4	**0.00 / 0.0**	0.50 / -0.4	**0.00 / 0.0**	0.00 / 0.6	0.00 / 0.1	0.00 / -0.3
DUU	**0.75 / 41.2**	0.00 / 0.3	0.15 / 11.6	0.00 / 0.0	**0.00 / 0.0**	0.50 / -0.5	**0.00 / 0.0**	0.00 / 0.0	0.00 / -0.1	0.00 / 0.0
kkunan	0.35 / 35.6	0.00 / 0.7	0.10 / 11.4	0.00 / 0.0	**0.00 / 0.0**	**0.50 / 0.4**	**0.00 / 0.0**	0.00 / 0.0	0.00 / -0.2	0.00 / -0.3
YOLOBOT	0.45 / 32.3	0.00 / -0.3	— / —	0.00 / 0.0	**0.00 / 0.0**	0.00 / 0.0	**0.00 / 0.0**	**0.10 / 1.0**	0.00 / 0.0	0.00 / -0.5

TABLE II
AGENT SCORE USING THE FORMULA-1 SCORE SYSTEM BASED ON AN AGENT'S AVERAGE POINTS PER GAME

Agent Name	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10	Total
Forward Model Approximation (FMA)	10	**25**	**25**	**25**	**25**	8	**25**	18	**25**	**25**	**211**
ercumentilhan	18	15	18	18	**25**	18	**25**	12	18	12	179
sampleRandom	6	18	15	15	**25**	12	**25**	15	10	18	159
sampleLearner	8	15	12	12	**25**	15	**25**	10	18	10	150
DontUnderestimateUchiha (DUU)	**25**	8	10	10	**25**	12	**25**	8	10	15	148
kkunan	12	10	8	10	**25**	**25**	**25**	8	6	10	139
YOLOBOT	15	6	6	10	**25**	6	**25**	**25**	12	6	136

sub-models, which are first trained on sample interactions with the game and later revised in case of contradictory observations. Using the extracted model we apply MCTS and BFS to find the best possible action at each game tick. The used search procedures were optimized by applying a state pruning, which reduces the number of evaluated states during the search phase.

We evaluated our approach in 10 games of the GVGAI competition. The developed agent overall outperforms previous algorithms. Our evaluation shows the enormous potential of forward model approximation. However, there is still much room for improvement, since the current version only considers the agents movement during future states. Complex interaction with other game elements are not yet modeled. Further improvements could be achieved by generalizing the prediction to many other attributes. Future work could focus on analyzing the capabilities of forward model approximation by increasing the number of predicted variables, while keeping the computational expense as low as possible.

Additional project files and the detailed evaluation can be found at: https://doi.org/10.17605/OSF.IO/VN3ZS

REFERENCES

[1] M. Genesereth, N. Love, and B. Pell, "General Game Playing: Overview of the AAAI Competition," *AI Magazin*, vol. 26, no. 2, pp. 62–72, 2005.

[2] D. Perez-Liebana, S. Samothrakis, J. Togelius, T. Schaul, and S. M. Lucas, "General video game ai: Competition, challenges and opportunities," in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence and the Twenty-Eighth Innovative Applications of Artificial Intelligence Conference*, D. Schuurmans and M. Wellman, Eds. Palo Alto, California: AAAI Press, 2016, pp. 4335–4337.

[3] D. Apeldoorn and G. Kern-Isberner, "Towards an understanding of what is learned: Extracting multi-abstraction-level knowledge from learning agents," in *Proceedings of the Thirtieth International Florida Artificial Intelligence Research Society Conference*, V. Rus and Z. Markov, Eds. Palo Alto, California: AAAI Press, 2017, pp. 764–767.

[4] T. Schaul, "A video game description language for model-based or interactive learning," *IEEE Conference on Computatonal Intelligence and Games, CIG*, 2013.

[5] D. Perez-Liebana, S. Samothrakis, J. Togelius, T. Schaul, and S. M. Lucas, "Analyzing the robustness of general video game playing agents," *IEEE Conference on Computatonal Intelligence and Games, CIG*, 2017.

[6] D. Perez Liebana, J. Dieskau, M. Hunermund, S. Mostaghim, and S. Lucas, "Open Loop Search for General Video Game Playing," *Proceedings of the 2015 on Genetic and Evolutionary Computation Conference - GECCO '15*, pp. 337–344, 2015.

[7] R. D. Gaina, J. Liu, S. M. Lucas, and D. Pérez-Liébana, "Analysis of vanilla rolling Horizon evolution parameters in general video game playing," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 10199 LNCS, pp. 418–434, 2017.

[8] E. İlhan and A. S. Etaner-Uyar, "Monte Carlo tree search with temporal-difference learning for general video game playing," in *2017 IEEE Conference on Computational Intelligence and Games (CIG)*. New York: IEEE, aug 2017, pp. 317–324.

[9] H. van Seijen, A. R. Mahmood, P. M. Pilarski, M. C. Machado, and R. S. Sutton, "True Online Temporal-Difference Learning," *Journal of Machine Learning Research (JMLR)*, vol. 17, pp. 1–40, dec 2015.

[10] D. Apeldoorn and G. Kern-Isberner, "When should learning agents switch to explicit knowledge?" in *GCAI 2016. 2nd Global Conference on Artificial Intelligence*, ser. EPiC Series in Computing, C. Benzmüller, G. Sutcliffe, and R. Rojas, Eds., vol. 41. EasyChair Publications, 2016, pp. 174–186.

[11] ——, "An agent-based learning approach for finding and exploiting heuristics in unknown environments," in *Proceedings of the Thirteenth International Symposium on Commonsense Reasoning, London, UK, November 6-8, 2017*, ser. CEUR Workshop Proceedings, A. S. Gordon, R. Miller, and G. Turán, Eds., vol. 2052. Aachen: CEUR-WS.org, 2018.

[12] L. Michael, "Causal Learnability," in *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011)*. Palo Alto, California: AAAI Press, 2011, pp. 1014–1020.

[13] A. Dockhorn, C. Doell, M. Hewelt, and R. Kruse, "A decision heuristic for Monte Carlo tree search doppelkopf agents," in *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, nov 2017, pp. 1–8. [Online]. Available: http://ieeexplore.ieee.org/document/8285181/

[14] A. Dockhorn, M. Frick, Ü. Akkaya, and R. Kruse, "Predicting Opponent Moves for Improving Hearthstone AI," in *17th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU)*, 2018, p. (to be published).

Applying Hybrid Reward Architecture to a Fighting Game AI

Yoshina Takano
Graduate School of Information Science and Engineering
Ritsumeikan University
Shiga, Japan
email is0263fs@ed.ritsumei.ac.jp

Wenwen Ouyang
College of Information Science and Engineering
Ritsumeikan University
Shiga, Japan
email is0444pk@ed.ritsumei.ac.jp

Suguru Ito
Graduate School of Information Science and Engineering
Ritsumeikan University
Shiga, Japan
email is0202iv@ed.ritsumei.ac.jp

Tomohiro Harada
College of Information Science and Engineering
Ritsumeikan University
Shiga, Japan
email harada@ci.ritsumei.ac.jp

Ruck Thawonmas
College of Information Science and Engineering
Ritsumeikan University
Shiga, Japan
email ruck@ci.ritsumei.ac.jp

***Abstract*—In this paper, we propose a method for implementing a competent fighting game AI using Hybrid Reward Architecture (HRA). In 2017, an AI using HRA developed by Seijen *et al.* achieved a perfect score of 999,990 in Ms. Pac-Man. HRA decomposes a reward function into multiple components and learns a separate value function for each component in the reward function. Due to reward decomposition, an optimal value function can be learned in the domain of Ms. Pac-Man. However, the number of actions in Ms. Pac-Man is only limited to four (Up, Down, Left, and Right), and till now whether HRA is also effective in other games with a larger number of actions is unclear. In this paper, we apply HRA and verify its effectiveness in a fighting game. For performance evaluation, we use FightingICE that has 40 actions and has been used as the game platform in the Fighting Game AI Competition at CIG since 2014. Our experimental results show that the proposed HRA AI, a new sample AI for the competition, is superior to non-HRA deep learning AIs and is competitive against other entries of the 2017 competition.**

***Index Terms*—deep learning, Hybrid Reward Architecture, muti-head agents, fighting game AI, FightingICE**

I. Introduction

Reinforcement learning (RL) successfully works in many games, but sometimes it is very slow and unstable to learn the optimal value function in other some games. Atari 2600 game Ms. Pac-Man was also one of the games where the optimal value function is difficult to learn, but Seijen *et al.* [1] applied Hybrid Reward Architecture (HRA) to their proposed game AI, which won a perfect score of 999,990. This result showed the effectiveness of HRA for RL. In HRA, a reward function is decomposed into multiple components, and HRA learns a value function for each component reward function. Because each component typically only depends on a subset of all the features, the overall value function is much smoother and can be easier approximated by a low-dimensional representation, enabling more effective learning. However, the number of actions in Ms. Pac-Man is only four (Up, Down, Left, and Right), and there has been no work showing whether HRA is also effective in other games with a large number of actions. We focus on fighting games, which in general have a much higher number of actions, apply HRA to a fighting game AI and propose a method for implementing a competent game AI.

Mnih *et al.* [2] achieved a big breakthrough by their Deep Q-Network (DQN). Their AI using DQN outperformed humans on a large number of Atari 2600 games, by learning a policy directly from pixels. Justesen *et al.* [3] showed how macromanagement decisions in StarCraft can be learned directly from game replays using deep learning. However, deep learning has not yet been successfully applied to fighting games in terms of performance. Yoon and Kim [4] applied DQN to a fighting game, but their result, against a non-moving opponent AI, only showed the potential of the DQN approach in this game genre. Nguyen [5] also applied a combination of RL and a Convolutional Neural Network, but the resulting AI was not able to defeat the champion of the Fighting Game AI Competition (FTGAIC) held at CIG 2015. Therefore, we consider it is worth examining if a competent fighting game AI can be implemented using HRA.

In this work, we use FightingICE which has been used as the platform in FTGAIC[1] at CIG since 2014 [6]. There are a number of existing studies using FightingICE in research as shown in the following.

Ishihara *et al.* [7] applied Monte-Carlo Tree Search (MCTS) to FightingICE and improved the AI performance by perform-

[1] http://www.ice.ci.ritsumei.ac.jp/˜ftgaic/
https://github.com/TeamFightingICE/FightingICE

978-1-5386-4359-4/18/$31.00 ©2018 IEEE

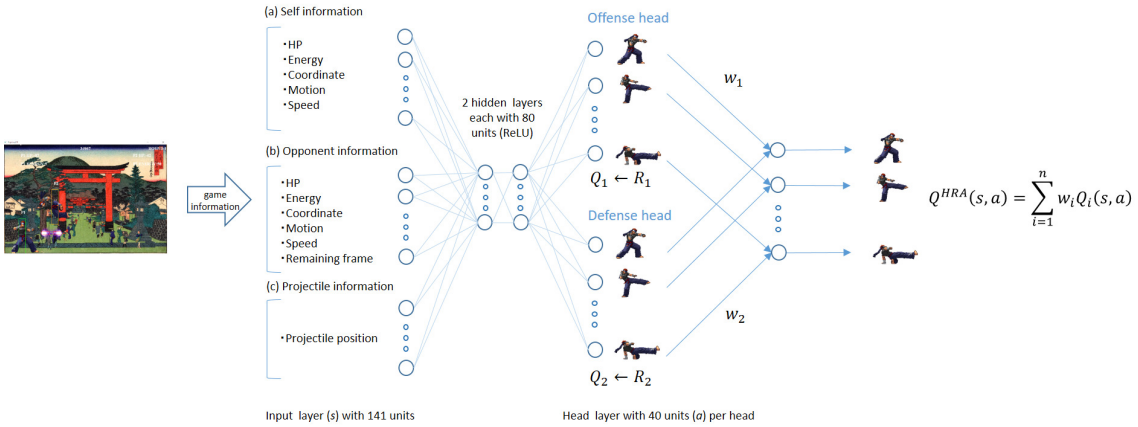


Fig. 2. HRA for FightingICE where the output layer combines the results from each of the two heads

Fig. 1. A screenshot of FightingICE

ing roulette selection in MCTS's play-out and introducing a set of rules to work with MCTS. Demediuk *et al.* [8] proposed and examined a variety of MCTS-based AIs whose aim is to adjust their difficulty level according to the player's level. Neufeld *et al.* [9] applied Hierarchical Task Network in their AI, an entry in the 2017 FTGAIC. However, their AI called HTNFighter came in 8th out of 10 entries in the competition. And more recently, Ishii *et al.* proposed a method for implementing a game AI with a persona using MCTS [10] while Ishihara *et al.* [11] not only considered difficulty adjustment but also believability in their game AI research where their AI was compared with an AI proposed by Demediuk *et al.* [8]. Figure 1 shows a screenshot of FightingICE.

II. Proposed Method

In this section, we describe a method for implementing a competent fighting game AI using HRA for FightingICE.

A. Hybrid Reward Architecture

We separate the reward function into n reward functions to learn the optimal Q-value function. The reward function of HRA is defined as follows:

$$R^{HRA}(s,a) = \sum_{i=1}^{n} w_i R_i(s,a), \quad \text{for all states } s, \text{ and actions } a, \tag{1}$$

where w_i is the weight of the each reward function. Each agent i of HRA, i.e., head i and its lower-layer structure, is trained using the respective reward function $R_i(s,a)$.

Because of individual reward function, each agent i, has its own Q-value function,. The Q-value function of HRA is defined as the weighted sum of all agents as follows:

$$Q^{HRA}(s,a;\theta) = \sum_{i=1}^{n} w_i Q_i(s,a;\theta), \tag{2}$$

where θ are training parameters. Each head can be viewed alternatively as a single DQN agent, and all the heads share multiple lower-level layers of DQN.

B. HRA for FightingICE

Figure 2 shows HRA for FightingICE. We propose using two heads: offense head and defense head. We call this multi-head AI. In FightingICE, to win the AI (self) has to give more damage to the opponent (opp) than the damage its receives. Thereby, we define the reward function as follows:

$$R_t^{HRA} = Reward_t^{offense} + Reward_t^{defense} \tag{3}$$

Here, we simply set the weight w of each head to 1. Each term on the right-hand side corresponds to the respective head and is described below where t and $t+1$ are the starting time of the current action and the subsequent action, respectively, by the AI.

1) Offense head: The offense head's role is to learn how to effectively give a damage to the opponent AI. The head obtains a reward when the AI gives a damage to the opponent AI. The reward function is defined as follows:

$$Reward_t^{offense} = HP_t^{opp} - HP_{t+1}^{opp}. \tag{4}$$

2) Defense head: The defense head's role is to learn how to effectively avoid opponent attacks. The head is penalized when the AI is hit by an opponent attack. The reward function is defined as follows:

$$Reward_t^{defense} = HP_{t+1}^{self} - HP_t^{self}. \tag{5}$$

C. Network Architecture

The network in this work is determined based on empirical results and consists of (1) an input layer with 141 units representing game information, (2) two hidden layers each with 80 units using the ReLU activation function, and (3) a head layer with two heads and 40 units per head, and (4) an output layer consisting of 40 units; in this network, the adjacent layers are fully connected. Because we can obtain the game information such as HP, Energy and coordinates of the AI and the opponent from FightingICE, we use 19 types of game-state information, leading to in total 141 input features. Note that game states can be reconstructed based on the 141 input features. FightingICE has 40 actions for controlling AIs, so each head in the head layer and the output layer have 40 units, each representing a different action. The aforementioned game information can be separated into self information, opponent information, and projectile information. We describe these features in detail below.

1) Self information: The self information consists of self HP, self Energy, self x coordinate, self y coordinate, self motion, self speed in the x direction, and self speed in the y direction; most of them are normalized to [0, 1]. However, only motion is represented as a one-hot vector of 56 units because the number of motions in FightingICE is 56.

2) Opponent information: Likewise, the opponent information consists of opponent HP, opponent Energy, opponent x coordinate, opponent y coordinate, opponent motion, opponent speed in the x direction, opponent speed in the y direction, and remaining frame (the number of remaining frames to complete the current action); accept for motion, they are normalized to [0, 1]. As done in the self information, motion is represented as a one-hot vector having 56 units.

3) Projectile information: The projectile information is composed of a projectile's x coordinate, y coordinate, and hit damage. Because there can be at most four projectiles at the same time, there are four sets of this information.

III. Experiments

In this section, we describe the details of FightingICE and the two experiments to verify the performance of the multi-head AI.

A. FightingICE

FightingICE is a real-time 2D fighting game platform. In this platform, one game consists of 60-second rounds, and one frame is set to 1/60 seconds. An AI has to decide and input its action in one frame. To make decision making by AIs challenging, they are given a game state by the system with a delay of 14 frames. Each of the two characters has two parameters: HP and Energy. Each character's initial HP is set to HP_{max} and will decrease when the respective character is hit; the initial Energy is set to 0. After 60 seconds elapse or HP of either character reaches 0, the game will proceed to the next round, and both characters' HP will be reset to HP_{max}. The character with the larger remaining HP at the end of round is the round winner. The value of HP_{max} is set to 10,000 in the first experiment to ensure the round length is fixed to 60 seconds and it is set to 400 in the second experiment according to the rule of Standard League of FTGAIC. The 2018 version of FightingICE is used in our experiments.

As described above, AIs cannot obtain game states with no delay in FightingICE. However, to construct tuples of state, action, reward, and next state used for DQN training, a mechanism is introduced, by which it is possible to determine whether an action of interest can be actually executed (in other words there are no other actions waiting to be executed) at a given – thus delayed – game state; if so, a tuple containing the action is created together with the other elements and added to a memory. We note that this mechanism is not used in any other parts in both experiments.

TABLE I
List of hyperparameters

Hyperparameter	Value
minibatch size	32
replay memory size	50000
target network update frequency	300
discount factor	0.9
learning rate	0.001
initial exploration	1
final exploration	0.1

B. Hyperparameters

We set the hyperparameters according to our empirical results. Their details are given in Table I.

C. Training against Machete

In the first experiment, we compare the proposed multi-head AI with a normal (single-head) DQN. We train both AIs for 1400 rounds against Machete that won the 2015 competition. Nguyen [5] also trained their AI against Machete, but their AI could not outperform Machete. Here, we are curious to see the performance of the multi-head AI against both single-head AI and Machete. We then analyze the learning process using the score, eqn. (6), over the number of training steps (rounds).

$$score_{self} = \frac{HP_{self}}{HP_{self} + HP_{opp}} \times a \tag{6}$$

where a is set to 1000, so the score above 500 indicates that the AI wins the opponent at the current round.

D. Comparisons with other AIs

In the second experiment, we compare the multi-head AI, the single-head AI, Machete, Ishihara's AI (ACEMctsAi) [7], and the AIs from the 2017 competition, including a sample MCTS AI (MctsAi). Here we focus on their performances in Standard League, a round-robin competition where there are two games – three rounds per game – switching sides for each pair of AIs, using a character called ZEN and its character data in the 2018 FTGAIC. Because a combo system introduced in the 2017 version has been removed, HTNFighter [9], utilizing the combo system, could not be run.

TABLE II
RESULT OF THE COMPETITION

AI name	# of wins	Win rate	Rank
GigaThunder	62	0.86	1st
MytNoAI	59	0.82	2nd
Mutagen	50	0.69	3rd
Multi-head AI	45	0.63	4th
FooAI	36	0.50	5th
Single-head AI	34.5	0.48	6th
Machete	34	0.47	7th
MegaBot	32.5	0.45	8th
MogakuMono	31	0.43	9th
ACEMctsAi	30	0.42	10th
MctsAi	26	0.36	11th
JayBot2017	19	0.26	12th
ZoneAI	9	0.13	13th

IV. Results and Discussions

In this section, we show the experimental results and our discussions in terms of whether the multi-head AI outperforms the single-head and other AIs.

A. Training against Machete

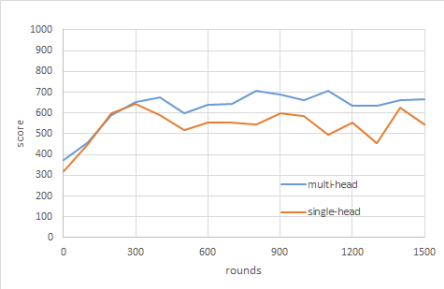


Fig. 3. Score Transitions against Machete

Figure 3 shows the average, sampled every 100 rounds, of five trials conducted in the first experiment. Both of the AIs using DQN start to defeat Machete from the 150th round, and this is the first time that DQN, to the best of our knowledge, was successfully applied in FightingICE and was able to defeat a former champion AI. The multi-head AI obtained the maximum score above 700, which outperforms the single-head. In addition, the score of the multi-head AI is more stable than that of the single-head one.

B. Comparisons with other AIs

The result of the competition is shown in Table II, where # of wins shows the number of winning rounds. The multi-head AI is the 4th among 13 AIs in Standard League. Video clips showing typical fights of the muti-head AI and the single-head AI in this competition are also available[2].

[2]http://www.ice.ci.ritsumei.ac.jp/˜ruck/HRA-cig2018.htm

V. Conclusions and Future Work

According to the experiment results, the multi-head AI obtained a higher score than the single-head AI in the training phase. In addition, the former was ranked higher than the latter in the conducted competition. The said results indicate that HRA is also effective in fighting games, where AIs conduct decision making to select their next actions from a large number of available actions. In this paper, we used only two heads. It is possible that using more heads will lead to a better performance. In addition, Machete was only used as the training opponent. The competition performance of the multi-head AI might be improved by switching the training opponents from a set of properly selected ones.

Acknowledgment

The authors would like to thank the anonymous reviewers for their helpful comments. They would also like to thank their lab members, in particular, the FightingICE team members for their fruitful discussions. This research was partially supported by Strategic Research Foundation Grant-aided Project for Private Universities (S1511026), Japan.

References

[1] H. Van Seijen, M. Fatemi, J. Romoff, R. Laroche, T. Barnes, and J. Tsang, *"Hybrid reward architecture for reinforcement learning,"* in Proc. Advances in Neural Information Processing Systems. 2017. pp. 5392-5402.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, *"Human-level control through deep reinforcement learning,"* in Nature, 2015, vol. 518, pp. 529-533.

[3] N. Justesen, and S. Risi, *"Learning Macromanagement in StarCraft from Replays using Deep Learning,"* in Proc. Computational Intelligence and Games (CIG), 2017 IEEE Conference on. IEEE, 2017. pp. 162-169.

[4] S. Yoon, and K.J. Kim, *"Deep Q Networks for Visual Fighting Game AI,"* in Proc. Computational Intelligence and Games (CIG), 2017 IEEE Conference on. IEEE, 2017. pp. 306-308.

[5] D.T.T. Nguyen, *Supervised and Reinforcement Learning for Fighting Game AIs using Deep Convolutional Neural Network,* in Master Thesis, Japan Advanced Institute of Science and Technology, Mar. 2017.

[6] F. Lu, K. Yamamoto, L. H. Nomura, S. Mizuno, Y. Lee and R. Thawonmas, *"Fighting Game Artificial Intelligence Competition Platform,"* in Proc. IEEE 2nd Global Conference on Consumer Electronics (GCCE), pp.320-323, 2013.

[7] M. Ishihara, T. Miyazaki, C.Y. Chu, T. Harada, R. Thawonmas, *"Applying and Improving Monte-Carlo Tree Search in a Fighting Game AI,"* in Proc. Proceedings of the 13th International Conference on Advances in Computer Entertainment Technology. ACM, 2016.

[8] S. Demediuk, M. Tamassia, W.L. Raffe, F. Zambetta, Xiaodong Li, *"Monte Carlo Tree Search Based Algorithms for Dynamic Difficulty Adjustment,"* in Proc. IEEE Conference on Computational Intelligence and Games (CIG 2017), New York City, USA, Aug. 22-25, 2017.

[9] X. Neufeld, S. Mostaghim, and D. Perez-Liebana, *"HTN Fighter: Planning in a Highly-Dynamic Game,"* in Proc. 2017 Computer Science and Electronic Engineering (CEEC 2017), Colchester, UK, pp. 189-194, Sep. 2017.

[10] R. Ishii, S. Ito, M. Ishihara, T. Harada, R. Thawonmas, *"Monte-Carlo Tree Search Implementation of Fighting Game AIs with Personas,"* in Proc. Computational Intelligence and Games (CIG), 2018 IEEE Conference on. IEEE, 2018.

[11] M. Ishihara, S. Ito, R. Ishii, T. Harada, R. Thawonmas, *"Monte-Carlo Tree Search for Implementation of Dynamic Difficulty Adjustment Fighting Game AIs Having Believable Behaviors,"* in Proc. Computational Intelligence and Games (CIG), 2018 IEEE Conference on. IEEE, 2018.

μCCG, a CCG-based Game-Playing Agent for μRTS

Pavan Kantharaju and Santiago Ontañón
Drexel University
Philadelphia, Pennsylvania, USA
pk398@drexel.edu, so367@drexel.edu

Christopher W. Geib
SIFT LLC
Minneapolis, Minnesota, USA
cgeib@sift.net

***Abstract*—This paper presents a Combinatory Categorial Grammar-based game playing agent called μCCG for the Real-Time Strategy testbed μRTS. The key problem that μCCG tries to address is that of adversarial planning in the very large search space of RTS games. In order to address this problem, we present a new hierarchical adversarial planning algorithm based on Combinatory Categorial Grammars (CCGs). The grammar used by our planner is automatically learned from sequences of actions taken from game replay data. We provide an empirical analysis of our agent against agents from the CIG 2017 μRTS competition using competition rules. μCCG represents the first complete agent to use a learned formal grammar representation of plans to adversarially plan in RTS games.**

***Index Terms*—adversarial planning, RTS games, combinatory categorial grammars**

I. Introduction

Real-Time Strategy (RTS) games are particularly useful in AI research because they provide a way to test AI that solve real world problems in a controlled environment. Since the call to research by Michael Buro in 2003 [1], RTS games have been used to solve challenging real-time AI problems such as decision making under uncertainty, resource management, opponent modeling, and adversarial planning. This work focuses on the problem of adversarial planning in deterministic and fully-observable RTS games by using Combinatory Categorial Grammars [2]. Combinatory Categorial Grammars (CCGs) are a well known grammar formalism developed for Natural Language Processing. Recent work by Geib [3] and Geib and Goldman [4] used probabilistic Combinatory Categorial Grammars to represent plans in a number of domains for the problem of plan recognition [5].

This work focuses on the domain of RTS games. Game tree search does not apply well to RTS games due to the enormous search space that needs to be traversed. Ontañón *et al.* describe a scenario where a 128x128 size game map with 400 units result in 16384^{400} possible game states [6]. Approaches to deal with this complexity in the literature range from hard-coding manually defined scripts (as is common in the StarCraft AI competition [6], to using abstraction in the action or state space [7]–[9], portfolio approaches [10], [11], or search strategies that attempt to scale up to the large branching factors of RTS games [12]. A promising approach to addressing this problem was work by Ontañón and Buro [13] who used an adversarial Hierarchical Task Network [14] (AHTN) planner to generate sequences of actions for playing μRTS. By integrating HTN planning with adversarial search, the advantages of domain-configurable planning, in terms of reduction of the search space, can be brought to RTS games. However, the HTN definitions used by the AHTN had to be hand authored.

This paper outlines two contributions. First, we present an alternative hierarchical planning formulation based on CCGs in the form of a μRTS agent called μCCG. Second, we show that we can learn a CCG plan representation from sequences of actions collected from game replay data. This is done by using a known CCG lexicon learning algorithm Lex_{Learn} by Geib and Kantharaju [15] to learn common sequences of action triples used by different μRTS agents. We limited ourselves to sequences of action triples because we wanted to model reactive behavior in RTS gameplay.

This paper is structured as follows. In section II, we provide a brief background of CCGs. In section III, we provide a brief overview of Lex_{Learn}. In section IV, we provide a description of μCCG and the hierarchical adversarial planner. In section V, we provide an empirical analysis of our agent against agents from the CIG 2017 μRTS tournament. In section VI, we provide a brief description of related work. Finally, in section VII we provide concluding remarks and future work.

II. Combinatory Categorial Grammars

This section briefly describes plan CCGs following the definitions used in the work of Geib *et al.* [3], [4]. Each action in a planning domain is associated with a set of CCG categories that can be thought of as functions and are defined recursively. We define a set of CCG categories $\mathcal{C}$ as follows.

Atomic categories are defined as a finite set of base categories denoted by $\{A, B, C...\} \in \mathcal{C}$. **Complex categories** are defined as $Z/\{W, X, Y, ...\} \in \mathcal{C}$ and $Z\backslash\{W, X, Y, ...\} \in \mathcal{C}$ where $\mathcal{C}$ is a set of categories, $Z \in \mathcal{C}$, $\{W, X, Y, ...\} \neq \emptyset$, and $\{W, X, Y, ...\} \in \mathcal{C}$. Atomic categories can be thought of as a zero-arity function that transitions from any initial state to a state associated with the atomic category. Complex categories define curried functions from states to states [16] based on the two left associative operators "\" and "/". These operators each take a set of **arguments** (the categories on the right hand side of the slash, $\{W, X, Y...\}$), and produces

978-1-5386-4359-4/18/$31.00 ©2018 IEEE

the state identified with the atomic category specified as its **result** (the category on the left hand side of the slash, Z). The slash also defines ordering constraints for plans, indicating where the category's arguments are to be found relative to the action. Forward slash categories find their arguments after the action, and backslash categories before it. We assume that all complex categories must be **leftward applicable** (all leftward arguments must be discharged before any rightward ones), and we only consider leftward applicable categories with atomic categories for arguments. A category R is the **root** or **root result** of a category G if it is the leftmost atomic result category in G. For example, for a complex category $(C\backslash\{A\})\backslash\{B\}$, the root would be C.

Using the above definition, a **plan lexicon** is a tuple, $\Lambda = \langle\Sigma, \mathcal{C}, f\rangle$, where Σ is a finite set of action types, $\mathcal{C}$ is a set of possible CCG categories, and f is a function such that $\forall a_i \in \Sigma : f(a_i) \rightarrow \{(c_{i,j} : \text{P}(c_{i,j}|a_i))\}$ such that $c_{i,j} \in \mathcal{C}$ and $\forall a_i, c_{i,j}, \sum_j \text{P}(c_{i,j}|a_i) = 1$. The function f maps each observable action, a_i, to a non-empty set of pairs $\{(c_{i,j} : \text{P}(c_{i,j}|a_i))\}$ each made up of a category, $c_{i,j}$, and the conditional probability that the action is assigned the category, $\text{P}(c_{i,j}|a_i)$. The definitions of actions and categories are extended to a first-order representation by introducing **parameters** for actions and atomic categories to represent domain objects and variables. The CCG learning algorithm Lex_{Learn} will learn a lexicon containing parameterized actions and categories, but the current version of the adversarial planner presented in this paper will not make use of any parameters during the planning process. We provide an explanation for this in Section IV. However, we still provide an example lexicon to illustrate the representation Lex_{Learn} generates:

$$f(\textbf{Train}(U_1, T)) \rightarrow \{\text{train}(U_1, T)) : 1\}$$
$$f(\textbf{Attack}(U_2, U_3)) \rightarrow$$
$$\{((\text{WrkRush})/\{\text{harvest}(U_4, R)\})\backslash\{\text{train}(U_1, T)\} : 1\}$$
$$f(\textbf{Harvest}(U_4, R)) \rightarrow \{\text{harvest}(U_4, R)) : 1\}$$

Note that **Train**(U_1, T), **Attack**(U_2, U_3), and **Harvest**(U_4, R) each have two parameters representing different units U_1, U_2, U_3, U_4, unit type T, and resource R. Since each action has only a single category, $P(c_{i,j}|a_i) = 1$. A full discussion of a CCG lexicon with parameterized actions and categories can be found in Geib [17].

III. Lex_{Learn}

This section briefly describes Geib and Kantharaju's CCG learning algorithm Lex_{Learn} [15]. Interested readers are referred to the full paper for more information. Lex_{Learn} is a supervised domain-independent CCG lexicon learning algorithm that generates lexicons for the ELEXIR framework, a CCG-based plan recognition algorithm [3]. Lex_{Learn} was built on prior work by Zettlemoyer and Collins [18] on CCG lexicon learning for Natural Language Processing (NLP). This is the first work that applies the learned CCG lexicon to the problem of adversarial hierarchical planning.

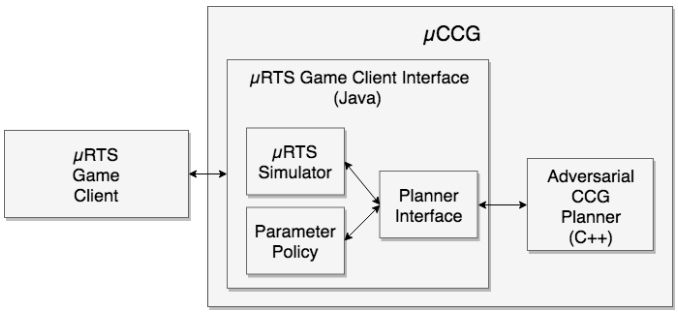


Fig. 1. μCCG Architecture Diagram

Lex_{Learn} takes as input an initial lexicon Λ_{init} and a training dataset, $\{(T_i, G_i) : i = 1...n\}$, where each T_i refers to a plan trace, a sequence of of observed actions that achieves a goal state G_i. Λ_{init} contains parameterized actions each paired with a single parameterized atomic category, and all actions in each T_i are contained in Λ_{init}. For each T_i, G_i pair in the training dataset, Lex_{Learn} incrementally updates Λ_{init} using two interleaved processes: category generation and parameter estimation.

Category generation is the process of generating new complex categories for actions in Λ_{init}. Given a plan trace T_i, the category generation process exhaustively enumerates the set of all possible categories for each action $a_j \in T_i$ using a set of predefined category templates. In the original work, pruning was used to limit the set of learned categories for actions that occurred more than once in T_i. However, in this work, we do not prune any generated complex categories for these actions. Λ_{init} is then updated by adding all constructed complex categories to action a_j.

The second process, parameter estimation, estimates the conditional probabilities $P(c_{j,k}|a_j)$ of all the action category pairs in the updated lexicon using stochastic gradient ascent [18]. Intuitively, each $P(c_{j,k}|a_j)$ represents a weighted frequency of how often the category $c_{j,k}$ is assigned to the action a_j during plan recognition. A full definition of the gradient used for gradient ascent can be found in the original paper [15].

IV. μCCG Agent

This section describes our μRTS game-playing agent, μCCG. Figure 1 provides an architecture diagram of our agent. There are two main components to μCCG as seen in the diagram: adversarial CCG planner and μRTS game client interface. The game client interface contains three subcomponents. The μRTS simulator is used to simulate a μRTS game state for planning, and the planner interface provides a bridge between the adversarial CCG planner and the μRTS environment. We describe the adversarial CCG planner and the parameter policy components from Figure 1 below.

At each game frame, the agent generates the best possible set of actions it can find for a given game state, given an allotted time. As per the CIG 2018 μRTS tournament rules, our agent is time constrained to 100ms per game frame. Our

procedure ACCG(s, T_+, T_-, t_+, t_-, d)
  if $\neg canIssueActions(s', +) \wedge d == MD$ **then**
    return
  end if
  $s' = simulateUntilNextChoicePoint(s)$
  if $GameEnd(s') \vee d \leq 0 \vee$ *both trees traversed* **then**
    return $(T_+, T_-, reward(s'))$
  end if
  if T_+ is traversed **then**
    Complete planning for min, ignore max
  end if
  if T_- is traversed **then**
    Complete planning for max, ignore min
  end if
  if $canIssueActions(s', +)$ **then**
    if $t_{-,c}$ is complex $\wedge$ root of $t_{+,c}$ is next **then**
      return $ACCG(\gamma(t_{+,a}, s'), T_+, T_-, t_+, t_-, d-1)$
    end if
    return $ACCGMax(s', T_+, T_-, t_+, t_-, d)$
  end if
  if $t_{-,c}$ is complex $\wedge$ root of $t_{-,c}$ is next **then**
    return $ACCG(\gamma(t_{-,a}, s'), T_+, T_-, t_+, t_-, d-1)$
  end if
  return $ACCGMin(s', T_+, T_-, t_+, t_-, d)$
end procedure
procedure ACCGMAX(s, T_+, T_-, t_+, t_-, d)
  if $t_{+,c}$ is atomic **then**
    $t = Update(T_+)$
    return $ACCG(\gamma(s, t_{+,a}), T_+, T_-, t, t_-, d-1)$
  end if
  $T_+^* = \perp, T_-^* = \perp, r^* = -\infty$
  if $t_+ == nil$ **then**
    $t_+ = T_+$
  end if
  $c_{next} = NextCat(t_{+,c})$
  $\mathcal{C} = AllDecompWithRoot(c_{next})$
  $\mathcal{C}'$ = N action, category pairs from $\mathcal{C}$ with highest UCB1 score
  for all $t \in \mathcal{C}'$ **do**
    $T_+' = AddToStack(t)$
    $(T_+', T_-', r') = ACCG(s, T_+', T_-, t, t_-, d)$
    if $r' > r^*$ **then**
      $T_+^* = T_+', T_-^* = T_-', r^* = r'$
    end if
  end for
  return (T_+^*, T_-^*, r^*)
end procedure

Fig. 2. Pseudocode for CCG Adversarial Planner

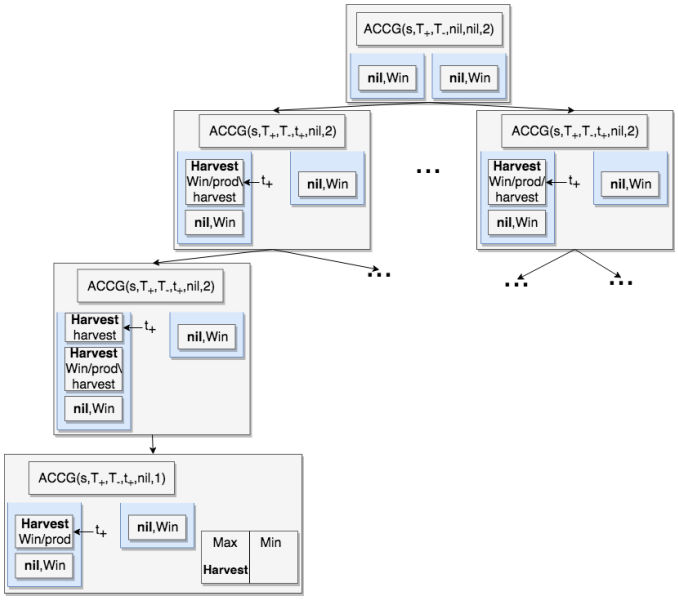


Fig. 3. Example Execution of Adversarial CCG Planner For Max Player

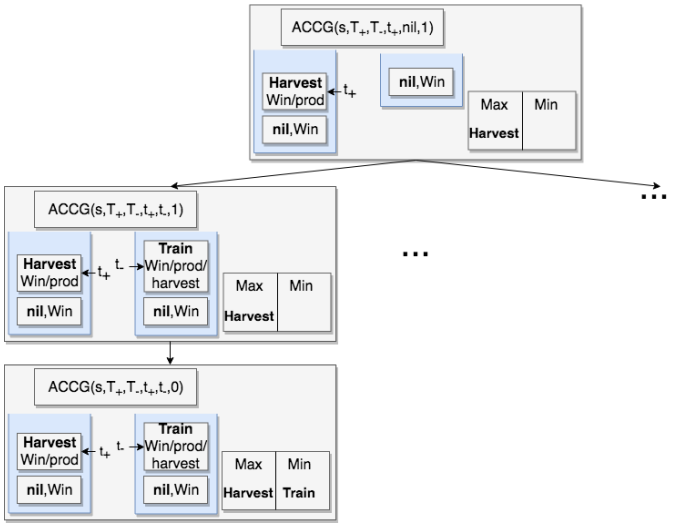


Fig. 4. Example Execution of Adversarial CCG Planner For Min Player

adversarial planner, motivated by Ontañón and Buro [13], uses a variant of minimax for RTS games.

Complex CCG categories represent a temporal relationship between states of a world. Given the complex category G/C\A, the sequence of actions resulting in state A must be completed and successful before executing the action(s) resulting in state G and state C. However, in the case of RTS games, actions can be executed in parallel in a given game state by multiple units. This requires us to relax the temporal restrictions defined by CCGs. For example, let A represent the state in which the agent executed the action **Train** for a base to train a worker unit and C represent the state in which the agent executed the **Attack** action for a heavy unit to attack an enemy unit. If both of these actions can be executed in a given game state, and **Train** fails to execute, that should not result in **Attack** failing as both actions are executed on different units. Therefore, during the planning process, if such a situation arises, instead of backtracking, the whole plan is still considered, and **Train** is replaced by an empty action.

In the current version of μCCG, our planner does not generate action parameters. Parameters for each action are generated using a hand-authored *Parameter Policy*. Although our planner can, in theory, generate action parameters, we don't currently use the planner to define parameters for actions because some of the parameters require information beyond what the state representation received by our planner currently contains, such as terrain and resources. For example, the attack command uses both unit and map layout, which the state

representation we currently provide to our planner ignores. Future work will look into augmenting the state representation with this information.

We define CCG decomposition as the process of expanding an atomic category c to a set of action category pairs whose root is the same as c. For example, if we have the category Win, a possible decomposition would be the action category pair (**Attack**, (Win/({harvest}))\{train}) (representing a plan where we first want to **Train**, then **Attack**, and then **Harvest**). It is possible the the number of decompositions for c might be very large (in this particular example, we might have a very large number of decompositions for how to Win in the learned lexicon). Thus, we select only a subset of N decompositions ($N = 5$ in our experiments). To select the subset, we currently use the Upper Confidence Bound (UCB1) [19] policy. In order to use UCB1, the planner keeps track of how many times each decomposition has been selected so far during planning, and the number of time c is decomposed (these counts are global for each player and are not reset throughout a whole game). We currently use the conditional probability $P(c_{j,k}|a_j)$ of an action category pair $(a_j, c_{j,k})$ from the lexicon as the reward value for UCB1. Notice that this probability never changes throughout the planning process, since it's only determined during category learning. Thus, the effect of UCB1 here is just to vary the subset of decompositions that are considered during planning, so that the planner does not always just consider the highest probability pairs. As part of our future work, we would like to use the actual achieved reward of each decomposition during gameplay instead, making our planner one step closer to a CCG-based MCTS planner, which would be the end goal.

Our planner generates plans using three actions from μRTS, **Attack**, **Harvest**, and **Return**, and a modified **Produce** action. The **Produce** action is modified to additionally include the unit being produced, such as **ProduceWorker** or **ProduceBarracks**, allowing the planner to determine build order instead of hard-coding it. This is done, since, as mentioned above, our planner does not currently generate action parameters. Thus, this encodes the unit type parameter to produce units without actually having any parameter. We assume the planner does not plan any movement actions. Initial experimentation resulted in our agent always choosing to move instead of any other action due to how frequent the movement action is in actual gameplay. Therefore, to prevent this, if **Attack**, **Harvest**, or **Return** need to move in order to succeed in a given game state, then a move action is issued.

Figure 2 provides pseudocode for our CCG adversarial planning algorithm. We provide pseudocode for only $ACCGMax$ in Figure 2, but $ACCGMin$ is a mirror definition. We refer to T_+ and T_- as the max and min player's **plan stack**. Action, category pairs are added to a plan stack based on when they are found during plan search. Figures 3 and 4 provide an example of plan stacks for both the max (left stack) and min (right stack) player. We define t_+ and t_- as pointers to the current decomposition in the stack, and d, MD, and s as current depth, maximum depth, and current game state.

There are four functions in the planner that interface with μRTS (seen in Figure 2). The first function $simulateUntilNextChoicePoint$ provides the μRTS framework with a game state s, and simulates until either player min or max can issue an action. However, given that the learned CCG lexicon cannot ensure that actions would be issued to all units in a given game state, it might be the case that even after exhausting plan search for the max player, max can still issue actions. This would result in the min player never getting a chance to perform search. To avoid this situation, we currently simulate the game state for one game frame before checking whether any player can issue an action to ensure that the game state advances and does not get stuck, and thus giving the planner a chance to search for actions for both players. This would not be an issue with hand-crafted lexicons that could be defined in such a way that this situation never arises, but our planner needs to be robust with respect to learned lexicons that do not necessarily ensure actions will be produced for all units in the game.

The next function $canIssueActions$ provides the μRTS framework with game state s and a player (either $+$, $-$, or ? referring to player max, min, or either) to determine if the player is ready to execute actions. If the player is ?, then player max is prioritized over player min. The next function γ applies a given action from a decomposition, $t_{+,a}$ or $t_{-,a}$, to a state s, returning the next game state. The final function $reward$ computes a reward using a reward function based on the given state.

The next four functions are specific to CCG adversarial planning. First, $NextCat$ takes a category c, and returns the leftmost atomic category that is not the root c_{left}. If c is atomic, then c is returned. For example, if c is Win/{prod}\harvest as seen in max's plan stack in Figure 3, the leftmost atomic category c_{left} would be harvest. Second, $AllDecompWithRoot$ decomposes c_{left} and returns the set of action category pairs that have c_{left} as a root ($\mathcal{C}$ in Figure 2). Third, $AddToStack$ pushes a given action category decomposition pair to a given stack.

The fourth and final function $Update$ propagates down the stack to find and return the next complex category to search. This function is called only when an action is generated by the planner. Until a complex category with at least one argument is found, the function pops off any atomic categories. If the function finds arguments from complex categories that have been fully reduced to a sequence of actions, it removes the argument and removes the category if the result of removal makes the category atomic.

Figures 3 and 4 provide a small example of the planner's execution tree where Figure 4 follows directly after Figure 3, so we first look at Figure 3. Each node in the execution tree represents a call to $ACCG$. At each game frame, the planner is given top-level goals which, in the case of the example, is Win for both the min and max player, and game state s. These top-level goals are added to the min and max plan stack T_- and T_+ as the action, category pair (**nil**,Win), and $ACCG(s, T_+, T_-, nil, nil, 2)$ is called.

Next, the planner simulates s using $simulateUntilNext-$

$ChoicePoint$ until at least one player can execute an action, and checks the available player using $canIssueActions$. In our example, this is the max player so the max player's function $ACCGMax$ is called. Since t_+ is nil, the planner sets this to the first action category pair in the stack (**nil**,Win). Win is already the leftmost atomic category, so $NextCat$ just returns Win. Next, using the function $AllDecompWithRoot$, the planner decomposes Win to two action category pairs, but we focus on one: (**Harvest**,Win/harvest\prod). Finally the planner adds the action category pair to max's plan stack, assigns the pointer t_+ to it and calls $ACCG(s, T_+, T_-, t_+, nil, 2)$.

Since the max player can still issue actions, the function $simulateUntilNextChoicePoint$ returns the next state and the planner calls $ACCGMax$ to decompose the complex category pointed to by t_+. In $ACCGMax$, $NextCat$ returns the leftmost atomic category harvest, and calls *AllDecompWithRoot* returning one decomposition, (**Harvest**, harvest). The planner adds this decomposition to the plan stack, points t_+ to it, and calls $ACCG(s, T_+, T_-, t_+, nil, 2)$. Next, $simulateUntilNextChoicePoint$ again returns the next game state, and calls $ACCGMax$ as the max player can still issue an action.

Calling *ACCGMax* again, the planner decomposes harvest to the action **Harvest**. The planner applies the action to the game state s, and calls *Update* to traverse down the stack. Since the planner already decomposed harvest, it pops the action category pair from the stack and its occurrence in the complex category Win/prod\harvest.

Next, as illustrated in Figure 4 the planner calls $ACCG(s, T_+, T_-, t_+, nil, 1)$. Note that the depth is only decremented when an action is issued by either player. After calling $simulateUntilNextChoicePoint$, planner detects that the max player can no longer issue actions, and the min player can. Similar to the max player, the planner calls $ACCGMin$ to decompose the min player's Win category into (**Train**,Win/prod/harvest), point t_- to it, and call $ACCG(s, T_+, T_-, t_+, t_-, 1)$.

Because this entire category is rightward-looking, the first leftmost category is actually the root, Win. This means that the next action to issue in the plan is **Train**. Therefore, instead of calling $ACCGMin$, the planner adds the action **Train** from the action category pair, and applies that action to the game state s. Finally, calling $ACCG(s, T_+, T_-, t_+, t_-, 0)$, completes planning.

Next, we look at the parameter policy, defined as follows. Any distance computation in our policy is computed using Euclidean distance. Given the **Attack** action, all produced offensive units are ordered to attack their closest enemy unit. We define offensive units as *Ranged*, *Heavy*, and *Light*. Worker units are not offensive because, from initial experiments, the agents would use the workers to attack instead of harvesting and building an army. Given the **Harvest** action, the agent finds the closest resource to a random base, and finds the closest worker to that resource. If the agent is within range of the resource, it harvests, and moves towards the resource if not. Given the **Return** action, the agent finds the closest base to a random worker. The agent then checks whether the worker is close to the base and moves if not.

The **Produce** action works differently. To prevent any resource contention when producing units, we only allow a single producing action to execute at a time. The planner dictates what units are constructed. However, in order to improve the game play strength of our agent for the competition setting, we impose some constraints on the planner output (this basically encodes our human domain knowledge of what a μRTS agent should do). The first restriction is that worker unit production is limited to a maximum of 2*(width of the map)/8 + 1 units to prevent over construction of workers in some maps. We wanted the agent to create at maximum three workers for the minimum map size of 8x8, and add two workers each time the game map quadrupled in size. We only used the width because most of the maps were squares. This limit will most definitely be changed for the competition. The second restriction is that we only allow a single worker unit to construct *Barracks* and *Base* to prevent the agent from constantly having to decide who should create these units. If the worker dies, another worker is chosen as the constructor.

V. Empirical Evaluation

The objective of our experiments is to test the effectiveness of a learned CCG lexicon and the adversarial CCG planner by evaluating it in the μRTS environment. In order to do this, we generated a dataset of plan traces from μRTS game replays using agents from last years's μRTS competition. Recall from Section III that a plan trace is defined as a sequence of observed actions. We then learned a CCG lexicon based on this dataset, and used it to play the game. We then evaluate game playing strength in the eight open maps that will be used for the 2018 μRTS competition, and compare against all the bots that participated in such competition.

We used the CCG lexicon learning algorithm Lex_{Learn} by Geib and Kantharaju [15] to generate a CCG lexicon for adversarial CCG planning. Lex_{Learn}'s parameters were tuned to the same values as Geib and Kantharaju's experiments. Recall from Section IV that the CCG adversarial planner only plans using four actions: **Attack**, **Harvest**, **Return**, and a modified **Produce** action. Below is the initial lexicon provided to Lex_{Learn}:

$$f(\textbf{Attack}) \rightarrow \{\text{attack} : 1\}$$
$$f(\textbf{Harvest}) \rightarrow \{\text{harvest} : 1\}$$
$$f(\textbf{Return}) \rightarrow \{\text{return} : 1\}$$
$$f(\textbf{ProduceBarracks}) \rightarrow \{\text{produce} : 1\}$$
$$f(\textbf{ProduceBase}) \rightarrow \{\text{produce} : 1\}$$
$$f(\textbf{ProduceWorker}) \rightarrow \{\text{produce} : 1\}$$
$$f(\textbf{ProduceLight}) \rightarrow \{\text{produce} : 1\}$$
$$f(\textbf{ProduceHeavy}) \rightarrow \{\text{produce} : 1\}$$
$$f(\textbf{ProduceRanged}) \rightarrow \{\text{produce} : 1\}$$

We note that **Attack**, **Harvest**, and **Return** have separate atomic categories, and each **Produce** action is given the same

TABLE I
TOURNAMENT RESULTS (WINS / LOSSES / TIES)

	POLightRush	POWorkerRush	RandomBiasedAI	NaiveMCTS	PVAIML_ED	StrategyTactics	μCCG	Total	Win Ratio
POLightRush	-	45-20-15	76-2-2	59-18-3	35-35-10	30-35-15	64-7-9	309-117-54	0.7000
POWorkerRush	20-45-15	-	71-0-9	32-39-9	35-35-10	31-38-11	64-10-6	253-167-60	0.5896
RandomBiasedAI	2-76-2	0-71-9	-	1-56-23	10-53-17	5-73-2	9-62-9	27-391-62	0.1208
NaiveMCTS	18-59-3	39-32-9	56-1-23	-	46-31-3	18-59-3	36-34-10	213-216-51	0.4969
PVAIML_ED	35-35-10	35-35-10	53-10-17	31-46-3	-	18-54-8	53-15-12	225-195-60	0.5313
StrategyTactics	35-30-15	38-31-11	73-5-2	59-18-3	54-18-8	-	70-1-9	329-103-48	0.7354
μCCG	7-64-9	10-64-6	62-9-9	34-36-10	15-53-12	1-70-9	-	129-296-55	0.3260

TABLE II
MAP RESULTS FOR μCCG (WINS / LOSSES / TIES)

Maps	POLightRush	POWorkerRush	RandomBiasedAI	NaiveMCTS	PVAIML_ED	StrategyTactics
FourBasesWorkers8x8	5-5-0	0-10-0	5-5-0	0-10-0	0-10-0	1-9-0
TwoBasesBarracks16x16	0-10-0	3-8-0	10-0-0	7-3-0	3-7-0	0-10-0
NoWhereToRun9x8	2-7-1	5-0-5	8-0-2	1-9-0	4-0-6	0-2-8
DoubleGame24x24	0-2-8	0-9-1	3-0-7	0-3-7	1-4-5	0-10-0
basesWorkers8x8A	0-10-0	0-10-0	6-4-0	1-9-0	7-2-1	0-10-0
basesWorkers16x16A	0-10-0	0-10-0	10-0-0	8-2-0	0-10-0	0-10-0
BWDistantResources32x32	0-10-0	1-9-0	10-0-0	7-0-3	0-10-0	0-9-1
(4)BloodBath.scmB	0-10-0	1-9-0	10-0-0	10-0-0	0-10-0	0-10-0

TABLE III
CIG 2018 μRTS TOURNAMENT MAPS AND GAME CYCLES

Maps	*Number of Game Cycles*
FourBasesWorkers8x8	3000
TwoBasesBarracks16x16	4000
NoWhereToRun9x8	3000
DoubleGame24x24	5000
basesWorkers8x8A	3000
basesWorkers16x16A	4000
BWDistantResources32x32	6000
(4)BloodBath.scmB	8000

atomic category. This relegated the decision of unit production to the adversarial planner. If each **Produce** action was given different atomic categories, then Lex_{Learn} would embed build information directly into the generated lexicon.

Our training dataset consists of plan traces derived from replay data of μRTS games. Specifically, we generated replay data by running a five-iteration Round Robin tournament on each of the open maps from the CIG 2018 μRTS tournament, shown in Table III with agents *POWorkerRush*, *POLightRush*, *PVAIML_ED*, and *StrategyTactics* [20], where each agent played as Player 1 and Player 2. Games on each map were limited to the number of game cycles stated in Table III. Next, we used the replay data to generate a set of 50,000 training instances, pruning any **Move** actions as we do not wish to learn any movement actions. While there were more training instances that could be generated (three action sequences with nine possible actions would require at minimum 729 instances to cover all possible permutations of three action sequences), we believe that 50,000 instances was enough for training. Each training instance corresponds to a 3-action behavior employed by the agents. The sequences of actions were limited to three to allow our agent to plan within the 100ms time limit as per the CIG 2018 tournament rules.

The adversarial CCG planner has three tunable parameters. The first parameter is the constant for UCB1, which was set to 20. The second parameter is the number of searched action category decompositions, N, which was set to 5. The third and final parameter is the maximum depth, which we set to 6 as that is the maximum number of actions that could be planned by both the max and min player in our adversarial planning search. These parameters were set to get results for the paper, but will be optimized for the competition.

We tested μCCG against six baseline agents: *RandomBiased*, *POWorkerRush*, *POLightRush*, *NaiveMCTS*, *StrategyTactics* [20], and *PVAIML_ED* using the eight open maps from the CIG 2018 μRTS tournament provided in Table III. We ran a five-iteration Round-Robin tournament where each agent played as both Player 1 and Player 2. Our experiments used all of the rules stated in the CIG 2018 tournament, except that we gave our agent a 30ms extra grace period per game frame to produce an action, since the purpose of these experiments was just to compare the agents.

Table I provides "Wins-Losses-Ties" and Win ratio (# wins + 0.5 $\times$ # ties) from our five-iteration Round-Robin tournament. Overall, our agent placed second-to-last in terms of Win ratio. Looking at Table II, which provides per-map results for μCCG, we were able to easily beat the RandomBiased agent, even winning more games than PVAIMIL_ED and NaiveMCTS. Additionally, we see that against the POWork-

erRush, RandomBiasedAI, and PVAIML_ED agents, μCCG didn't lose a single match on the NoWhereToRun9x8 map.

μCCG did come close to outperforming NaiveMCTS, with a win-loss difference of two. Table II indicates that for the first four maps, NaiveMCTS significantly outperformed our agent. However, for the last three maps μCCG outperformed NaiveMCTS. Specifically for the last two maps, μCCG never lost a single match against NaiveMCTS. Additionally, μCCG was able to win a few games against the two top performing agents in the CIG 2017 μRTS competition, StrategyTactics, and POLightRush. We believe that with an improved parameter policy, μCCG could potentially win more games against these agents.

We believe that μCCG may have won on the last two maps against NaiveMCTS and RandomBiased because the maps were relatively large with (4)BloodBath.scmB being the largest map in the set at 64x64. As the size of the map gets larger and the number of units increases, NaiveMCTS has to search a larger search space. We believe that this search space explosion resulted in NaiveMCTS losing games against μCCG. We believe that the ties between μCCG and NaiveMCTS in BWDistantResources32x32 may have been due to the game reaching the maximum number of cycles. μCCG may have destroyed most of NaiveMCTS' units, but as a result caused NaiveMCTS to start playing optimally because the state space decreased.

We believe most of our losses were due to a few factors related to the parameter policy and planning. First, we speculate that μCCG may have delayed constructing barracks and offensive units because we only allowed a single unit to construct units at a time. Thus, if μCCG was creating workers, it wouldn't be able to construct barracks or any offensive units. Second, we believe that not allowing workers to attack enemy units may have caused μCCG to lose on small maps. On small maps, μCCG would not have time to construct offensive units and a group of offensive workers would immediately overwhelm us. Third and finally, we believe that coupling the attack action with build order planning may have stopped offensive units from attacking. Offensive units could only attack if the attack action was administered by the agent. If the planner didn't generate an attack action, all offensive units would stop attacking (even mid-assault on the enemy).

Although μCCG still does not outperform state of the art bots, our experiments show that the idea of using an adversarial CCG planner with a learned CCG lexicon generated from a domain-independent CCG lexicon learning algorithm is viable for RTS games. As part of our future work before the 2018 competition, we would like to optimize our parameter policy, as well as the training set and planning algorithms to maximize game-play performance, which was not a priority at this point.

VI. Related Work

There are several areas of research that are closely related to our work: 1) RTS game-playing agents, 2) Adversarial Planning, and 3) Plan Learning. There is a plethora of work in the scientific literature on creating agents to play RTS games such as Starcraft and μRTS. Synnaeve and Bessière present BroodwarBotQ which uses a Bayesian model for unit control in Starcraft [21]. Uriarte presents Nova, a Starcraft agent that combines several techniques used to solve different AI problems [22]. Churchill and Buro present UAlbertaBot which optimizes build order planning using action abstractions and heuristics [23]. Other Starcraft agents include Skynet and Berkeley's Overmind [24].

There is also a large amount of prior research on adversarial planning, but we state a few here. Stanescu *et al.* present an approach to hierarchical adversarial search motivated by the chain of command employed by the military. Specifically, they employ game tree search on two layers of plan abstractions [25]. Willmott *et al.* [26] presents GoBI, an adversarial HTN planner for the game of Go that uses α-β search with a standard HTN planner. α uses the HTN planner to generate an action, and passes the game state to β to generate their action while attempting to force α to backtrack its search. In recent years, Ontañón and Buro used Hierarchical Task Networks (HTNs) [13] and minimax to adversarially plan against an opponent in RTS games. This work builds off this, but uses CCGs instead of HTNs for planning, and learns the plan representation instead of hand-authoring one.

The last ares of related research is plan learning. Hogg *et al.* present HTN-Maker which learn HTN methods from analyzing the state of the world before and after a given sequence of actions [27]. Zhuo *et al.* [28] present HTNLearn which builds an HTN from partially-observable plan traces. Nejati *et al.* [29] present a technique for learning a specialized class of HTNs from expert traces. Finally, Li *et al.* [30] present a learning algorithm that successfully learns probabilistic HTNs using techniques from probabilistic Context-Free Grammar induction. The two main differences of our work is that we learn a plan CCG representation and we learn this for an RTS domain.

VII. Conclusion

This paper presents initial work on a CCG-based game playing agent for μRTS called μCCG. This paper provides two main contributions. First, we presented an alternative hierarchical planning formulation based on CCGs. Second, we show that we can learn a CCG plan representation from sequences of actions collected from game replay data. We also provided initial results of μCCG against against other μRTS agents. Our results seem promising and demonstrate that μCCG can use a learned representation generated by a domain-independent learning algorithm to play against other agents. We are currently in the process of improving the agent for the CIG 2018 μRTS tournament, specifically the parameter policy.

There are a few directions for future work. First, we want to look into interweaving other RTS problems such as terrain analysis and resource management into the planner to improve the planning process. Second, we want to look into improving hierarchical learning of CCGs by incorporating RTS domain knowledge into the learning process as Lex_{Learn}

is a general CCG plan learning algorithm. Third, once our agent is able compete against other μRTS agents, we want to apply our CCG adversarial planner to the commercial RTS game, Starcraft. Fourth, we trained Lex_{Learn} on sequences of three actions due to the time constraint defined in the μRTS tournament rules, but plans can be larger than three actions. Thus, we want to learn from larger sequences of actions. Finally, for the current version of our planner, we relaxed the temporal restrictions of CCG lexicons in order to accommodate parallel actions. However, we would like to investigate this issued further and design a general framework to deal with concurrent actions in the context of CCGs.

REFERENCES

[1] M. Buro, "Real-time strategy gaines: A new ai research challenge," in *Proceedings International Joint Conference in Artifical Intelligence*, 2003, p. 15341535.

[2] M. Steedman, *The Syntactic Process*. Cambridge, MA, USA: MIT Press, 2001.

[3] C. W. Geib, "Delaying commitment in plan recognition using combinatory categorial grammars," in *Proceedings of the 21st International Jont Conference on Artifical Intelligence*, ser. IJCAI'09. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2009, pp. 1702–1707. [Online]. Available: http://dl.acm.org/citation.cfm?id=1661445.1661719

[4] C. W. Geib and R. P. Goldman, "Recognizing plans with loops represented in a lexicalized grammar," in *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*, ser. AAAI'11. Palo Alto, California, USA: AAAI Press, 2011, pp. 958–963. [Online]. Available: http://dl.acm.org/citation.cfm?id=2900423.2900576

[5] C. F. Schmidt, N. Sridharan, and J. L. Goodson, "The plan recognition problem: An intersection of psychology and artificial intelligence," *Artificial Intelligence*, vol. 11, no. 1-2, pp. 45–83, 1978.

[6] S. Ontañón, G. Synnaeve, A. Uriarte, F. Richoux, D. Churchill, and M. Preuss, "A survey of real-time strategy game AI research and competition in StarCraft," *IEEE Transactions on Computational Intelligence and AI in Games (TCIAIG)*, vol. 5, pp. 1–19, 2013.

[7] R.-K. Balla and A. Fern, "UCT for tactical assault planning in real-time strategy games," in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI 2009)*, 2009, pp. 40–45.

[8] N. Justesen, B. Tillman, J. Togelius, and S. Risi, "Script-and cluster-based UCT for StarCraft," in *Computational Intelligence and Games (CIG), 2014 IEEE Conference on*. IEEE, 2014, pp. 1–8.

[9] A. Uriarte and S. Ontañón, "Game-tree search over high-level game states in RTS games," in *Proceedings of the AAAI Artificial Intelligence and Interactive Digital Entertainment conference (AIIDE 2014)*, 2014.

[10] D. Churchill and M. Buro, "Portfolio greedy search and simulation for large-scale combat in StarCraft," in *IEEE Conference on Computational Intelligence in Games (CIG 2013)*, 2013, pp. 1–8.

[11] M. Chung, M. Buro, and J. Schaeffer, "Monte carlo planning in RTS games," in *Proceedings of the IEEE Conference on Computational Intelligence in Games (CIG 2005)*, 2005.

[12] S. Ontanón, "Combinatorial multi-armed bandits for real-time strategy games," *Journal of Artificial Intelligence Research*, vol. 58, pp. 665–702, 2017.

[13] S. Ontanón and M. Buro, "Adversarial hierarchical-task network planning for complex real-time games," in *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[14] K. Erol, J. Hendler, and D. S. Nau, "UMCP: A sound and complete procedure for hierarchical task network planning," in *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems (AIPS 94)*, 1994, pp. 249–254.

[15] C. Geib and P. Kantharaju, "Learning combinatory categorial grammars for plan recognition," in *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*, 2017.

[16] H. Curry, *Foundations of Mathematical Logic*. Dover Publications Inc., 1977.

[17] C. W. Geib, "Lexicalized reasoning about actions," *Advances in Cognitive Systems*, vol. Volume 4, pp. 187–206, 2016.

[18] L. S. Zettlemoyer and M. Collins, "Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars," in *UAI '05, Proceedings of the 21st Conference in Uncertainty in Artificial Intelligence, Edinburgh, Scotland, July 26-29, 2005*, ser. UAI'05. AUAI Press, 2005, pp. 658–666.

[19] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine learning*, vol. 47, no. 2-3, pp. 235–256, 2002.

[20] N. A. Barriga, M. Stanescu, and M. Buro, "Combining strategic learning and tactical search in real-time strategy games," in *Proceedings of the Thirteenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE-17)*, 2017, pp. 9–15.

[21] G. Synnaeve and P. Bessi?e, "A bayesian model for rts units control applied to starcraft," in *2011 IEEE Conference on Computational Intelligence and Games (CIG'11)*, Aug 2011, pp. 190–196.

[22] A. Uriarte, "Multi-reactive planning for real-time strategy games," Master's thesis, Universitat Autònoma de Barcelona, 01 2011.

[23] D. Churchill and M. Buro, "Build order optimization in starcraft." in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2011, pp. 14–19.

[24] M. Buro and D. Churchill, "Real-time strategy game competitions," *AI Magazine*, vol. 33, no. 3, p. 106, 2012.

[25] M. Stanescu, N. Barriga, and M. Buro, "Hierarchical adversarial search applied to real-time strategy games," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2014.

[26] S. Willmott, J. Richardson, A. Bundy, and J. Levine, "An adversarial planning approach to go," in *International Conference on Computers and Games*. Springer, 1998, pp. 93–112.

[27] C. Hogg, H. Muñoz Avila, and U. Kuter, "Htn-maker: Learning htns with minimal additional knowledge engineering required," in *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 2*, ser. AAAI'08. AAAI Press, 2008, pp. 950–956. [Online]. Available: http://dl.acm.org/citation.cfm?id=1620163.1620220

[28] H. H. Zhuo, H. Muñoz-Avila, and Q. Yang, "Learning hierarchical task network domains from partially observed plan traces," *Artificial Intelligence*, vol. 212, pp. 134 – 157, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0004370214000447

[29] N. Nejati, P. Langley, and T. Konik, "Learning hierarchical task networks by observation," in *Proceedings of the 23rd international conference on Machine learning*. ACM, 2006, pp. 665–672.

[30] N. Li, W. Cushing, S. Kambhampati, and S. Yoon, "Learning probabilistic hierarchical task networks as probabilistic context-free grammars to capture user preferences," *ACM Trans. Intell. Syst. Technol.*, vol. 5, no. 2, pp. 29:1–29:32, Apr. 2014. [Online]. Available: http://doi.acm.org/10.1145/2589481

Improving Hearthstone AI by Combining MCTS and Supervised Learning Algorithms

Maciej Świechowski*
*Silver Bullet Labs
Liwiecka 25, Warsaw, Poland
m.swiechowski@mini.pw.edu.pl

Tomasz Tajmajer*†
†Institute of Informatics, University of Warsaw
Banacha 2, Warsaw, Poland
t.tajmajer@mimuw.edu.pl

Andrzej Janusz†‡
‡eSensei
Mazowiecka 11/49, Warsaw, Poland
janusza@mimuw.edu.pl

***Abstract*—We investigate the impact of supervised prediction models on the strength and efficiency of artificial agents that use the Monte-Carlo Tree Search (MCTS) algorithm to play a popular video game *Hearthstone: Heroes of Warcraft*. We overview our custom implementation of the MCTS that is well-suited for games with partially hidden information and random effects. We also describe experiments which we designed to quantify the performance of our Hearthstone agent's decision making. We show that even simple neural networks can be trained and successfully used for the evaluation of game states. Moreover, we demonstrate that by providing a guidance to the game state search heuristic, it is possible to substantially improve the win rate, and at the same time reduce the required computations.**

***Index Terms*—MCTS, Hearthstone, machine learning, neural networks, heuristic**

I. Introduction

Hearthstone: Heroes of Warcraft is a free-to-play online video game developed and published by Blizzard Entertainment. Its simple rules and appealing design made this game successful among casual players. According to Blizzard's data, in 2017 the player-base of the game was about 70 million and it grows with each of the released expansions. The game is also popular within the eSport community, with cash-prize tournaments and many international events every year.

Hearthstone is an example of a turn-based collectible card game. During the game, two players choose their hero with a unique power and compose a deck of thirty cards. They spend mana points to cast spells, weapons and summon minions to attack the opponent, with the goal to reduce the opponent's health to zero or below. Due to a large number of distinct cards which implement various game mechanics, and special in-game effects which often have randomized outcomes, Hearthstone is an example of a game where actions may have non-deterministic results. Moreover, during a game each player is unaware of cards that the opponent holds in hand, nor the ordering of yet-to-be-drawn cards in his deck. Finally, since a player may perform several actions in each turn of the game and ordering of those actions is pivotal to player's success, Hearthstone features great combinatoric complexity. All the above properties make Hearthstone a demanding challenge for AI-controlled bots that are designed to play this game. One objective of this article is to explain how our implementation of the Monte Carlo Tree Search (MCTS) algorithm deals with those problems. We also aim to discuss the means by which MCTS can be facilitated by machine learning algorithms and provide experimental evaluation of its performance.

The paper is organized as follows. In the next section, we continue with providing context of the research and show related initiatives. In Section III, the MCTS algorithm is discussed with the focus on problems encountered in Hearthstone such as randomness, hidden information and combinatorial complexity. We also shed some light on the game simulator used for this research. The subsequent section is devoted to methods of combining MCTS with machine-learning-based heuristics. Finally, the last two sections contain a description of empirical experiments which we conducted to evaluate our Hearthstone agents and conclusions, respectively.

This research was co-funded by the Smart Growth Operational Programme 2014-2020, financed by the European Regional Development Fund under a GameINN project POIR.01.02.00-00-0150/16, operated by The National Centre for Research and Development (NCBiR), and by the Silver Bullet Labs company.

II. Related work

In recent years, Hearthstone has become a testbed for AI research. A community of passionate players and developers have started the HearthSim project (hearthsim.info) and created several applications that allow simulating the game for the purpose of AI and machine learning experiments. A few spin-offs of that project, e.g., MetaStats (metastats.net), provide tools for the players, which facilitate gathering data from their games. These portals obtain and aggregate users' data, such as game results, deck compositions, card usage statistics and provide this information to the community.

Several groups of researchers from the field of machine learning and AI have already chosen Hearthstone for their studies. In [1], authors used evolutionary algorithms to tackle the problem of building good decks. They used the results of simulated games performed by simple AI bots as fitness function values. Even though this study was described by the authors as preliminary, the developed method was able to construct reasonable decks from a basic set of cards. However, one drawback of this method is the fact that it strongly depends on the performance of the AI bots used for the evaluation of the decks.

A few research groups were also considering a problem of constructing an artificial agent able to play Hearthstone.

In particular, [2] used Monte-Carlo Tree Search (MCTS) algorithm to choose an optimal action policy in the game. Furthermore, [3] used deep neural networks to improve performance of a MCTS-based Hearthstone bot, called Silverfish. The combination of MCTS with prediction models make those approaches similar to early versions of DeepMind's AlphaGo program [4]. It is worth noticing, however, that unlike Go, in Hearthstone players do not have full information about the game state and many actions have non-deterministic outcomes. These two properties make this game much more challenging for the game state tree search algorithms, such as MCTS [5].

There were also attempts at constructing models for predicting cards that are likely to be played by an opponent during a game. For instance, in [6] the author used data from 45,000 Hearthstone games to extract sequences of played cards and represent each record as a bag of card bi-grams. By investigating co-occurrence probabilities, the method described in that study was able to correctly predict opponent's card which will most likely appear during the following turns of the game, in over 50% of cases. Such a high predictability can be explained by the fact that even though the number of possible Hearthstone decks is enormous, players tend to build their decks in accordance to certain archetypes and their composition is often inspired by the decks used by other influential players.

Hearthstone was also a topic of international data mining competitions. The first one, *AAIA'17 Data Mining Challenge: Helping AI to Play Hearthstone*[1], was focused on developing a scoring model for predicting win chances of a player, based on detailed description of a single game state [7]. Although the data in this competition was generated using very simple bots which were choosing their moves at random, the best models created by participants were able to achieve AUC scores above 0.80. The winner used an ensemble of 1-dimensional convolutional neural networks to extract features from each combination of both players' cards on the board [8]. A year later, the second edition of this challenge was launched. The task in *AAIA'18 Data Mining Challenge* was to predict win-rates of Hearthstone decks, based on a history of match-ups between AI bots playing with similar decks.

Various other card games were also studied in the literature related to machine learning and AI. For instance, in [9] authors consider heads-up no-limit poker as an example of a game with hidden information. They describe a DeepStack algorithm which aims to handle the information asymmetry between players by combining recursive reasoning with learning from self-played games. As a different example one can give the game Magic: The Gathering, studied, e.g., in [10]. Due to the notable similarity to Hearthstone, these games pose many similar challenges. In our work, however, we focus only on Hearthstone. The growing interest of the machine learning community in applications related to video games stems from the fact that solutions to many game-related problems could be easily transfered to real-life issues, such as planning [11], real-time decision making [12], [13] and, ultimately, general AI.

[1]Competition's web page: https://knowledgepit.fedcsis.org/contest/view.php?id=120

III. Playing Hearthstone with Monte-Carlo Tree Search

A. Game Simulator

The access to a game simulator allows game-playing agents to perform dynamic reasoning about the game. The idea is to run separate simulations that do not affect the actual (main) state of the played game. This is a reason why a simulator is often called a "forward model" as it enables forward planning. Its performance, i.e., how many states it can visit per second, is crucial for all methods that are based on searching the space of the game such as MCTS, min-max or MTD(f). Therefore, we have written a simulator for Hearthstone with the aim of achieving the highest run-time performance. The main features of our simulator are: (1) written entirely in C++ for high-performance, (2) it performs 10K full games per second, in average, and 30K when limiting to basic cards only, (4) makes big use of inheritance and polymorphism (e.g., Secret : Spell : Card), (5) effects such as hero powers are modeled as (non-collectible) cards, (6) the total number of implemented cards = 483, (7) the implemented cards allow for making staple decks from the standard meta-game.

The simulator calculates legal moves in each state of the game, updates the state after a move is chosen, tests whether the game reached a terminal state and calculates scores in a finished game. States and actions are comparable and hashable. We have divided complex game actions into atomic simple actions, e.g., when the "SI-7 Agent" card is played, up to three simple actions are generated: (1) Choose a card from your hand (SI-7 Agent), (2) Choose a target on the battle-field, where the minion is about to be placed, (3) Choose a target for the battle-cry: deal 2 damage, provided that the required *combo* condition was met. Similarly, an attack move consists of two simple actions - choosing a character, which will attack and choosing a target to attack.

B. Monte Carlo Tree Search

Monte-Carlo Tree Search (MCTS) [14] has become the state-of-the-art algorithm for game tree search. It is the algorithm to go in domains such as Go [15], Hex [16], Arimaa [17], General Game Playing (GGP) [18] or General Video Game Playing (GVGP) [19]. This technique is a natural candidate for universal domains such as GGP or GVGP, because given only the way (interface) to simulate games, the same implementation of MCTS will work for any game. It has also been increasingly successful in board games such as Settlers of Catan [20] or 7 Wonders [21].

In essence, the MCTS is a combination of three ideas: storing statistics in the game tree, random sampling by means of simulations to gather statistics and the Upper Confidence Bounds method to select nodes based on the statistics gathered so far. The *Upper Confidence Bounds applied to Trees (UCT)* addresses the exploitation-exploration problem and it

is a generalization of the Upper Confidence Bounds (UCB-1) method. The UCT formula is as follows:

$$a^* = \arg\max_{a \in A(s)} \left\{ Q(s,a) + C\sqrt{\frac{ln\,[N(s)]}{N(s,a)}} \right\} \quad (1)$$

where $A(s)$ is a set of actions available in state s, $Q(s,a)$ denotes the average result of playing action a in state s in the simulations performed so far, $N(s)$ - a number of times state s has been visited in previous simulations and $N(s,a)$ - a number of times action a has been sampled in this state in previous simulations. Constant C controls the balance between exploration and exploitation. It has to be tuned, but provided that scores of games are confined to the $[0,1]$ interval, the sensible starting value is $\sqrt{2}$.

The algorithm typically consists of four phases: selection, expansion, simulation and backpropagation. Algorithms (1) and (2) describe the usage of these phases.

(1) Selection. Traverse the nodes, that are already stored in the tree. At each level, the next node is chosen according to the selection policy - the UCT method, by default.

(2) Expansion. A certain number of new nodes is added to the tree. In the classical MCTS variant, only one node is added by each iteration, which is a good trade-off between the algorithm's efficiency and memory usage.

(3) Simulation. Starting from the last visited state in the tree, play (simulate) the game till the end. No nodes are added to the tree in this phase. Actions for each player are chosen randomly, however, there are extensions of the MCTS algorithm that introduce heuristics in the simulation. This phase is also called "Monte-Carlo phase".

(4) Back-propagation. Starting from the last visited node in the tree, which is the one the simulation started from, all the way up to the root node, update the $Q(s,a)$ values based on the result of the simulation.

1) Handling Imperfect Information: The majority of successful applications of the MCTS algorithm have been done in the realm of perfect information games, i.e., games in which each player has complete information about the current state of the game. Games with hidden information have been proven to be difficult for any combinatorial method such as game-tree search. There have been many variants and extensions to the MCTS proposed to deal with imperfect information. However, they can be clustered into two types of approaches:

1) **Perfect Information Monte Carlo Tree Search (PIMC)** - this method determines (guesses) all information that is hidden and, from that point, treats the game as perfect information one. Variants of PIMC differ in the way how many distinct determinizations they perform and how the knowledge obtained from running the algorithm with different determinizations is combined. The two major problems related to PIMC [22] are strategy fusion and nonlocality [23].
2) **Information Set Monte Carlo Tree Search (ISMCTS)** [23] - this variant uses the concept of information sets, which are abstract groups of states that are indistinguishable from a particular player's perspective. In ISMCTS, a node in the game tree is associated with an information set rather than a single state. Therefore, the decisions of a player are made based upon what the player actually observes. ISMCTS is much less susceptible to the problems of strategy fusion and nonlocality. However, ISMCTS is typically much harder to implement as it requires to simulate games under imperfect information or deal with partially observable moves.

We propose an algorithm, which is a combination of ISMCTS and PIMC. From the first concept, we borrow the idea of information sets. However, they are not used to simulate games under hidden information. Instead, they serve as keys in the so-called transposition table. The transposition tables are a way to model the "game-tree" without duplicated nodes, which would occur if there is more than one way to reach the same state. The "tree" effectively then becomes a directed acyclic graph (DAG). Transposition tables are also often used to combine symmetric states in order to reuse calculations. In the transposition table we used, the values are nodes and there is a unique key-value mapping between information sets and nodes. Each node contains a hashmap of edges with key being a player's move. Each edge contains the statistics of the particular move and a pointer to the next node as observed in the current iteration of MCTS. The next node pointer might vary in subsequent iterations if the same move can have multiple outcomes (non-determinism) and thus lead to various information sets. From the PIMC concept, we borrow the idea of determinizations. At the beginning of each MCTS iteration, a copy of a hidden information state is determined into a perfect information state. This is not to be confused with information set. The default solution to determinization is to sample the state randomly among possible legal states. However, when generating games for machine learning experiments, we used the "cheater" approach that can determinize the correct state. Such an approach is often used in teaching sessions. In particular, in card games, human experts teach beginners how to play with open cards. In our case, the justification is that the "cheater" allows for generating stronger games quicker.

In our implementation, there are two interfaces for the concept of the game state:

Game state for simulations (*GS*) - this is the only interface used to apply the logic of the game such as determining legal moves, applying moves, checking whether the game has ended or getting the result of the game. This interface is used both in the selection and simulation phases. However, in the selection, the other interface (information sets) is used as well.

Information Set Game state for statistics *(IS*) - this is an abstraction of a state with possible hidden information. It represents all kind of information, based on which a player will take actions. The idea is to use only a subset of the simulation game state in order to group states. Such a separate interface not only allows for ignoring hidden information but also for

reducing the resolution of the state. For instance, states that are similar in terms of some arbitrary measure can be grouped together. The information sets in our approach are plain data storage objects. The only methods the *IS* interface contains are *hash* and *equals*, what enables efficient equality comparisons.

After the *GS* has been determined, the selection phase starts from the root node. In each visited node during that phase, the set of currently legal moves is computed and intersected with the set of all moves observed in the node so far. Each move is associated with an edge. Active edges are the ones that correspond to moves that are currently available. The active edges are scored according to the selection formula (c.f. Equation 1) and the best scored edge is chosen. Next, the *GS* interface is used to apply the selected edge's move and compute the resulting state. This state is then used to generate an information set. We call this process capturing the information set and the *GS* requires an implementation of the *capture()* method that returns the *IS* from a given player's perspective. The perspective is decided based on which player is active in the current state. Once the *IS* is created, it is used to query the transposition table for the next node to traverse. If no such node exists, it is added to the transposition table with the key equal to the current *IS* and the selection phase is terminated. The selection phase is repeated for the next node until the termination condition (a node visited for the first time) is not satisfied. Because nodes are matched with information sets, this statistics of actions performed within the same *IS* are clustered together. Moreover, this allows to significantly reduce the combinatorial size of the game tree in comparison with using regular game states as nodes. When the selection phase ends, the last seen *GS* is passed to the simulation phase as the starting state. The result of a simulation is propagated to all edges chosen in the selection phase.

Algorithm 1 Pseudocode of the main MCTS loop.
The $simulation$ method starts from the $movingState$ and performs a quasi-random simulation and returns the result of the game. It can be replaced by another evaluation procedure as discussed later in the paper.

```
1:  procedure ITERATE(state)
2:      rootNode ← createRoot(state)
3:      node ← rootNode                          ▷ current node
4:      while elapsedTime < allotedTime do
5:          movingState ← determinize(state)
6:          while mcts.selection ≠ finished do
7:              if movingState.terminal ≠ true then
8:                  node ← node.select(movingState)
9:              end if
10:         end while
11:         propagate(simulation(movingState))
12:     end while
13: end procedure
```

2) Handling Randomness: Non-determinism in games can quickly increase the combinatorial complexity to enormous levels. For example, there are $5.36 * 10^{28}$ different deals possible in the game of Bridge. Randomness is also prevalent in Hearthstone, with effects such as "discover a random spell" or "deal from X to Y damage". Each unique random outcome would most likely result in a different state, and therefore, would require its own node in the tree.

Algorithm 2 Pseudocode of the inner MCTS loop. The $findOrCreate$ method accepts an information set and returns the corresponding node from the transposition table.

```
1:  procedure NODE.SELECT(movingState)
2:      moves ← movingState.getMoves()
3:      currentEdges ← []
4:      for each move in moves do
5:          edge ← allEdges[move]
6:          if edge not found then
7:              edge ← new edge(move)
8:              allEdges[move] ← edge
9:          end if
10:         edge.N ← +1                  ▷ incr. observed count
11:         currentEdges.push(edge)
12:     end for
13:     chosenEdge ← selection(currentEdges)          ▷ UCT
14:     chosenMove ← chosenEdge.getMove()
15:     chosenEdge.V ← +1                  ▷ incr. visit count
16:     if chosenEdge.V == 1 then
17:         mcts.selection ← finished
18:     end if
19:     movingState.apply(chosenMove)
20:     is ← capture(movingState)                  ▷ create IS
21:     tt ← mcts.getTranspositionTable()
22:     chosenEdge.nextNode ← tt.findOrCreate(is)
23:     return chosenEdge.nextNode
24: end procedure
```

The novelty of our MCTS implementation is complete exclusion of nature moves. This makes the game modeling and simulating significantly easier using our library. Actions may include any non-determinism. This is possible, because we do not store game-states directly in the tree as results of actions. As shown on Algorithm (2), each time a move is played, we compute the resulting state dynamically, even if the move has been already sampled in previous iterations. The resulting state is used to create the information set, which then is used to fetch the next node to visit. In consequence, statistics of moves are averaged according to the probability distribution of various random effects. If a move is good in average, the score will be high and it will be chosen more frequently in the selection phase of the MCTS algorithm.

3) Handling Combinatorial Explosion: We have already introduced the idea of the separation of "virtual game states" modeled as Information Sets and the regular game states for simulations. This allowed us to gather statistics in a much more coarse-grained representation of state-space. However, the combinatorial complexity of the game is still very high due to the number of possible attacks, the fact that attacks can

be done in chosen order and the options to intertwine playing cards between the attacks. The authors of [24] have calculated that, in the pessimistic case, there are approximately 10^{10} possible ways of performing the attacks. Quite often, however, lots of permutations of attacks will result in the same state in the end and there is no need to examine all of them. To tackle this problem, we have developed the so-called "board solver" - a heuristic that generates a sequence of attack actions in a given state. In general, the heuristic first checks if it can kill the opponent in one turn and does it if possible. If not, the heuristic will check whether the opponent is likely to win during their next turn and if so, the attacks will focus on killing the most threatening opponent minions. If no of these cases appear, the heuristic will score all possible single attacks based on the $gain - loss$ of the board potential. A single attack is a pair (attacker, defender). In Hearthstone, there are at most 8 attackers and 8 defenders, so, in the pessimistic case, 64 scores need to be calculated. The attacks are applied in a greedy fashion, i.e., the best scored attack is applied first (if possible), next the second best and the process continues until there are no more legal attacks. An application of an attack may render some of the following attacks illegal, for example when they use an attacker that has already attacked or defender that has already been killed. The heuristic for attacks is used as an artificial action in the game: "use solver". The MCTS is allowed to choose this action at any point during the turn, but only once per turn. Once the action is chosen, the attack moves are generated and applied, to there will not be any attacks move anymore during the turn for the minions that are already on the board.

4) Interfacing heuristics with MCTS: The MCTS algorithm is quite powerful on its own, but it can still benefit from domain-specific optimizations. It has been proven that, in more complex games such as Go [4] with huge branching factor and delayed rewards of taking actions, the vanilla method needs to be enhanced by some form of heuristics.

This weakness has motivated us to combine this algorithm with heuristics represented by prediction models. Such prediction models can be trained to either predict the outcome of the game by looking at a potential next state (candidate state) of the game or at a potential action (candidate action). In the scope of this paper, we will use the terms "machine learning prediction models" and "heuristic evaluation" interchangeably.

There is a couple of ways to combine external heuristics with the MCTS algorithm. The authors of paper [25] give a nice review of four common methods: Tree Policy Bias, Simulation Policy Bias, Early Cutoff and Move Ordering. We use the first three of them:

(1) Tree Policy Bias - here the heuristic evaluation function is included together with the $Q(s, a)$ in the UCT formula (see Eq. 1) or its equivalent. A typical implementation of this idea is called *Progressive Bias* [26], in which the standard UCT evaluation is linearly combined with the heuristic evaluation with the weight proportional to the number of simulations. The more simulations are performed, the more statistical confidence, and therefore, the higher weight is assigned to the standard UCT formula.

(2) Simulation Policy Bias - here the heuristic values affect probabilities of certain actions in the simulation phase to make simulated players stronger and, therefore, each simulation a better approximation of a potential future game. The two most common implementations are pseudo-roulette selection with probabilities computed using Boltzmann distribution (where the heuristic evaluation is used) or the so-called epsilon-greedy approach [27]. In the latter, the action with the highest heuristic evaluation is chosen with the probability of ϵ or a random one with the probability of $1 - \epsilon$.

(3) Early Cutoff - terminate the simulation earlier (e.g., with some probability or at fixed depth) and return the heuristic evaluation of the last reached state instead of the terminal one. In [25], this enhancement is reported to achieve the best results among the tested methods.

The aforementioned AlphaGo program employs both, Tree Policy Bias and Simulation Policy Bias. Motivated by its success, we decided to apply a similar approach for Hearthstone.

IV. Augmenting MCTS with Machine Learning

The state of the art implementations of MCTS, such as AlphaZero, use deep neural networks for providing heuristic evaluations of states and actions. Two main approaches are used – so called *value network* is a deep neural network that provides the predictions of a game outcome given a state of the game. The predictions are usually provided as scores which can be interpreted as probabilities of winning the game by each player. Such predictions may be used by MCTS to foresee an outcome of a playout without simulating it until the terminal state, or even to entirely replace the simulation phase. A *policy network* is another type of a neural network that given the state of a game provides values of each action available in that state. Policy network may thus provide information about which actions should be chosen in a state. As shown in [4], [28], the use of value and policy network heuristics significantly improves the performance of MCTS methods, enabling them to beat humans in very complex games.

In our solution we will focus on the value network heuristic for Hearthstone. We will use an iterative approach to neural network training, which uses large amount of hearthstone games, generated by self-playing bots.

A. Game-state vectorization with embeddings

Heuristic functions for evaluating game states require a vectorized representation of the state. It is common to use hand-crafted attributes to represent particular aspects of the state and then, using some weighted combination of those attributes, derive a value representing the utility of a state. While this approach works for games such as chess, it may be difficult to engineer such attributes for much more complex games such as Go or Hearthstone. As we use deep learning methods for obtaining heuristic functions, it is possible to represent Hearthstone states by large vectors composed of values of low-level features such as: attributes of each minion

on the board (HP, attack, taunt, charge etc.), attributes of each player (HP, weapons, mana, hero type, etc.), attributes of cards in hand (type, mana cost, etc.) and general attributes (turn number, cards in deck, etc.). Moreover, as most cards in Hearthstone have custom descriptions that define special effects, it is necessary to extend the vectors by meaningful representations of particular cards.

One way to represent the cards in a relatively low-dimensional vector space is by using a word2vec model [29] to learn the embeddings from cards' textual descriptions. It can be done either by aggregating vector representations of words from the texts or by training a paragraph vector model [30], where each paragraph corresponds to a single card. Since descriptions of Hearthstone cards are relatively short and use a limited vocabulary, it is expected that a dimensionality of our embeddings should be much lower than in other common applications of the word2vec model. We experimentally checked that using more than 16 dimensions brings negligible improvements, and thus we used embedding size 10 in our further experiments. To learn the embeddings, we used the skip-gram model implemented in TensorFlow. Apart from the embedding size, standard parameter values were used, i.e. context size was set to 10 and the batch size was 256. The model was trained for 300 epochs using a stochastic gradient descent optimizer, with a learning rate 0.1, decreased by a factor of 10^{-1} after every 100 epochs.

In our final solution, we used a vectorizer that had 750 elements, including all low-level features for both players and utilized embeddings to represent all cards and minions.

B. State evaluation with value network

Our state evaluation heuristic uses a fully connected neural network for providing the win probabilities of each player. The network consists of three dense layers with 256, 128 and 64 neurons respectively and uses *tanh* activation function. The input is a vector of size 750 (as described in the previous section), while the output consists of two neurons with a *softmax* activation. The network thus solves a classification task: given a state predict the winner.

The training data for the network is generated by recording games played between bots. During a simulation, the state of the game is vectorized to vector $\vec{S}$ at each step, and the final score of the game is stored as a two-element vector: $\vec{score} = [p1_{score}, p2_{score}]$. Next, the vectorized states are sampled randomly with some probability p and pairs $[\vec{S}, \vec{score}]$ are added to the training dataset. Random sampling is required, as consecutive states are highly correlated. Finally the network is trained to provide score given a state vector. We used ADAM optimizer with learning rate = 0.001

Value networks are trained to predict scores of games that were played with different decks as well as from the perspective of any of the two players. However, the accuracy of the predictions are better if there are separate networks trained for particular decks and even for particular player positions (first or second player).

In our preliminary tests we created a dataset with over 3.5M samples from games played by strong MCTS bots (cheater MCTS with 1 second per move) playing with 400 different decks. The network were trained to predict outcomes of the games played with any of the available decks and for any of the players. The accuracy of the value network trained using this dataset was evaluated on a separate validation set and reached 0.76.

We have used the trained value network for early termination of random simulations. The termination was done after the last move of a player in turn, but not earlier than after k=20 steps. After termination, the statistics in MCTS tree were updated with probabilities of winning obtained from the value network.

C. Iterative learning - mastering Hearthstone

To further improve the performance of our solution, we have prepared an environment for continuous, iterated learning of our machine learning models. The main idea is that MCTS with a heuristic may be used to generate games of progressively better quality. Those games may then be used to create more accurate heuristics, which may be used to generate games of even better quality. This process may be repeated many times for better optimization of the heuristics.

In our approach to iterative learning, we have started with plain MCTS to generate over 20000 games. Next, those games were used to generate an initial dataset consisting of randomly selected states and corresponding scores. Models for value networks were trained and used to generate the next version of the bot. Then, in each iteration, the bot played 3000 games, from which new state-score pairs were sampled and added to the training dataset. The training dataset length was clipped to 1M samples, so that after a few iterations older samples were removed and most recent samples were appended as in a FIFO buffer. The state-score pairs were sampled with probability $p = 0.5$. In each iteration, value networks were retrained from scratch using 80% of the training dataset. Remaining 20% was used for validation of the network.

Using iterated learning, we were able to achieve an accuracy of 0.775 for the first player and 0.794 for the second player, when training for one type of deck only. In the next section we describe in details the performance of particular bots.

V. Experiments

We have conducted a series of experiments to measure the skill of various Hearthstone bots based on MCTS and different heuristics. Due to the high complexity of Hearthstone, mainly caused by the large number of possible decks and the impact of random effects on the game outcome, we have restricted our test cases to only two decks: *ZooWarlock* and *CubeWarlock*. Moreover, we have fixed the positions of both players, so that *ZooWarlock* deck was always played by the first player, while *CubeWarlock* by the second.

In order to obtain the best possible version of the value network, we have run iterative training for 64 iterations. Next, we have created a hearthstone bot for each version of the

TABLE I: Evaluation results - 0.5 second per move

P1	P1 wins	P2 wins	P1 win %	P2 win %	P2
mcts	735	265	73,5%	26,5%	mcts
mctsVS	500	0	100,0%		random
mctsVS	391	108	78,4%		mcts
mctsV	410	90	82,0%		mcts
mctsS	395	105	79,0%		mcts
random	0	500		100,0%	mctsVS
mcts	219	280		56,1%	mctsVS
mcts	249	251		50,2%	mctsV
mcts	266	234		46,8%	mctsS

TABLE II: Evaluation results - 1 second per move

P1	P1 wins	P2 wins	P1 win %	P2 win %	P2
mcts	705	294	70,6%	29,4%	mcts
mctsVS	500	0	100,0%		random
mctsVS	364	135	72,9%		mcts
mctsV	380	120	76,0%		mcts
mctsS	358	143	71,6%		mcts
random	0	500		100,0%	mctsVS
mcts	224	276		55,2%	mctsVS
mcts	220	279		55,9%	mctsV
mcts	263	236		47,3%	mctsS

value network obtained during the iterative learning. Finally, we have used 64 versions of the bot to play over 50k matches between themselves and assigned a glicko2 rating [31] to each bot. Based on the glicko2 rating, we have selected the best bot, and thus the best value network, for the first and second player (obtained from 21st and 33rd iteration respectively).

For our final evaluation, we have compared plain MCTS (denoted by *mcts*) with two different heuristics: a) previously selected, best value networks from iterative learning - denoted by *V*; b) board solver described in section III-B3 - denoted by *S*. We have measured the impact of the value network, board solver and both of those combined together. Each configuration of the bot was used to play 500 games against plain MCTS bot. Moreover, we have also compared our solution with a randomly playing bot. To have a baseline for the performance, a 500-game match between only plain MCTS bots was played as well. The games were played with two time limits per move used by MCTS: 0.5 and 1.0 second. The results are presented in tables I and II. The strength of each bot is measured by the percentages of won games.

The baseline win-rates are 73.5% for the first player and 26.5% for the second in case of 0.5 second per move time limit. Increasing the time limit improves the strength of the second player, resulting in win-rates 70.6% for the first player and 29.4% for the second. The evaluation results show that each heuristic has a noticeable impact on the strength of the bot. As the first player has already a high win-rate, adding heuristics improves the win-rate by up to 9 percentage points. However, in case of the second player, adding heuristics may even double the win-rate.

TABLE III: A summary of results obtained in games between AI agents and human opponents.

P1	P1 wins	P2 wins	P1 win %	P2 win %	P2
Regular	7	7		50%	mctsVS-1s
Legend	12	9		43%	mctsVS-1s
mctsVS-1s	9	6	60%		Regular
mctsVS-1s	3	15	17%		Legend

It is important to note here that the type of deck used has a huge impact on the strength of the bot. The deck used by the first player has an aggressive, but fairly straightforward, style of play. The deck used by the second player, has on the other hand, a lot of complex strategies and needs to be played carefully; yet used by a skillful player, it has a much greater winning potential compared to the first deck. This fact may help to understand why the strength of the second player is increased so dramatically when using well-crafted heuristics.

Moreover, heuristics provide a larger advantage, when playing with lower time per move limit as MCTS performs a fewer number of iterations. A combination of a value network and board solver, when only 0.5 seconds per move are available for the MCTS to perform simulations, provide the greatest boost to the bot's strength. With 1 second per move available, the difference between using only value network and the combination of value network and board solver is minimal.

Finally, we have arranged matches between a few hearthstone players and our bot. The results are presented in table III. Games were played by two regular players (Hearthstone rank > 15, which is held by approx. 75% players) and two players with a Legendary rank (the best one with less than 0.5% of players).

VI. Conclusions

In this paper, a fully-fledged approach to constructing a Hearthstone playing bot was presented. Some novel features of the approach include modification of the MCTS algorithm to handle randomness without explicitly defined nature moves, a combination of the PIMC and ISMCTS methods to tackle imperfect information, and a heuristic solver for calculating attacks in Monte Carlo simulations. In addition, we designed and conducted machine learning experiments aimed at learning game state evaluation functions. Finally, an iterative learning loop aimed at creating the "ultimate bot" was proposed.

We can conclude that the resulting agent is likely to be among the strongest Hearthstone bots at the moment. Although Hearthstone has become a testbed for AI, there has not been yet proposed any universal benchmarking methods, so it is difficult to assess the strength other than by human observation, self-play between various versions of the agent or a random player. However, in all cases, the proposed solution shows its upper hand. The bot is able to win, with an impressive consistency, 100% games against the random player. It is also capable of winning games against Legend rank players, which

alone can be regarded as very promising. The human players reported that in many situations they felt the bot played really well. Finally, we have shown the progressive improvement of the bot's skills by sparing it against previous versions. We designated two decks for this experiment, but the approach can be generalized for any number of decks easily, e.g., as an ensemble that chooses the right model (or even blends a few of them) for the deck on the fly.

In order to benchmark our agent against other Hearthstone bots, we plan to submit it to the 2018 Hearthstone AI Competition held under the CIG (Computational Intelligence in Games) conference. Our submission to this competition will differ with the approach described in this paper in several details. It will work with the SabberStone (https://github.com/HearthSim/SabberStone) simulation engine as this is the official engine to be used during the competition. This simulator is only able to simulate approximately 200 games per second, on a modern high-end consumer PC, whereas our simulator performs 10000 games, on average. Because of this fact, we choose to limit the depth of the Monte Carlo simulations to the end of a single turn. At the end of the turn, the state evaluation function powered by machine learning will be used. We hope that the solutions adopted for the CIG competition will help us in designing even more cunning artificial Hearthstone agent, and as a consequence, move us one step further in the pursuit of the Grail of video games – smarter and challenging AI.

REFERENCES

[1] P. García-Sánchez, A. Tonda, G. Squillero, A. Mora, and J. J. Merelo, "Evolutionary deckbuilding in hearthstone," in *Computational Intelligence and Games (CIG), 2016 IEEE Conference on*. IEEE, 2016, pp. 1–8.

[2] A. Santos, P. A. Santos, and F. S. Melo, "Monte carlo tree search experiments in hearthstone," in *IEEE Conference on Computational Intelligence and Games, CIG 2017, New York, NY, USA, August 22-25, 2017*, 2017, pp. 272–279. [Online]. Available: https://doi.org/10.1109/CIG.2017.8080446

[3] S. Zhang and M. Buro, "Improving hearthstone AI by learning high-level rollout policies and bucketing chance node events," in *IEEE Conference on Computational Intelligence and Games, CIG 2017, New York, NY, USA, August 22-25, 2017*, 2017, pp. 309–316. [Online]. Available: https://doi.org/10.1109/CIG.2017.8080452

[4] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[5] P. I. Cowling, E. J. Powley, and D. Whitehouse, "Information set monte carlo tree search," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 2, pp. 120–143, June 2012.

[6] E. Bursztein, "I am a legend: Hacking Hearthstone using statistical learning methods," in *IEEE Conference on Computational Intelligence and Games, CIG 2016, Santorini, Greece, September 20-23, 2016*, 2016, pp. 1–8. [Online]. Available: https://doi.org/10.1109/CIG.2016.7860416

[7] A. Janusz, T. Tajmajer, and M. Świechowski, "Helping AI to play Hearthstone: AAIA'17 data mining challenge," in *Proceedings of the 2017 Federated Conference on Computer Science and Information Systems, FedCSIS 2017, Prague, Czech Republic, September 3-6, 2017.*, 2017, pp. 121–125. [Online]. Available: https://doi.org/10.15439/2017F573

[8] Ł. Grad, "Helping AI to play Hearthstone using neural networks," in *Proceedings of the 2017 Federated Conference on Computer Science and Information Systems, FedCSIS 2017, Prague, Czech Republic, September 3-6, 2017.*, 2017, pp. 131–134. [Online]. Available: https://doi.org/10.15439/2017F561

[9] M. Moravčík, M. Schmid, N. Burch, V. Lisý, D. Morrill, N. Bard, T. Davis, K. Waugh, M. Johanson, and M. Bowling, "Deepstack: Expert-level artificial intelligence in heads-up no-limit poker," vol. 356, no. 6337, pp. 508–513, 2017.

[10] P. I. Cowling, C. D. Ward, and E. J. Powley, "Ensemble determinization in monte carlo tree search for the imperfect information card game magic: The gathering." *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 4, pp. 241–257, 2012.

[11] R. Munos *et al.*, "From bandits to monte-carlo tree search: The optimistic principle applied to optimization and planning," *Foundations and Trends® in Machine Learning*, vol. 7, no. 1, pp. 1–129, 2014.

[12] D. Lee, "Game theory and neural basis of social decision making," *Nature neuroscience*, vol. 11, no. 4, p. 404, 2008.

[13] M. Buro and T. Furtak, "Rts games as test-bed for real-time ai research," in *Proceedings of the 7th Joint Conference on Information Science (JCIS 2003)*, vol. 2003, 2003, pp. 481–484.

[14] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A Survey of Monte Carlo Tree Search Methods," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, pp. 1–43, 2012.

[15] S. Gelly, L. Kocsis, M. Schoenauer, M. Sebag, D. Silver, C. Szepesvári, and O. Teytaud, "The grand challenge of computer go: Monte carlo tree search and extensions," *Commun. ACM*, vol. 55, no. 3, pp. 106–113, Mar. 2012. [Online]. Available: http://doi.acm.org/10.1145/2093548.2093574

[16] B. Arneson, R. B. Hayward, and P. Henderson, "Monte Carlo Tree Search in Hex," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 2, no. 4, pp. 251–258, 2010.

[17] O. Syed and A. Syed, *Arimaa - A New Game Designed to be Difficult for Computers*. Institute for Knowledge and Agent Technology, 2003, vol. 26, no. 2.

[18] M. R. Genesereth, N. Love, and B. Pell, "General Game Playing: Overview of the AAAI Competition," *AI Magazine*, vol. 26, no. 2, pp. 62–72, 2005.

[19] J. Levine, C. B. Congdon, M. Ebner, G. Kendall, S. M. Lucas, R. Miikkulainen, T. Schaul, and T. Thompson, "General Video Game Playing," *Dagstuhl Follow-Ups*, vol. 6, 2013.

[20] I. Szita, G. Chaslot, and P. Spronck, "Monte-carlo tree search in settlers of catan," in *Advances in Computer Games*. Springer, 2009, pp. 21–32.

[21] D. Robilliard, C. Fonlupt, and F. Teytaud, "Monte-carlo tree search for the game of 7 wonders," in *Workshop on Computer Games*. Springer, 2014, pp. 64–77.

[22] J. R. Long, N. R. Sturtevant, M. Buro, and T. Furtak, "Understanding the success of perfect information monte carlo sampling in game tree search." in *AAAI*, 2010.

[23] P. I. Cowling, E. J. Powley, and D. Whitehouse, "Information set monte carlo tree search," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 2, pp. 120–143, 2012.

[24] M. H. Andersson and H. H. Hesselberg, "Programming a hearthstone agent using monte carlo tree search," Master's thesis, NTNU, 2016.

[25] K. Waledzik and J. Mańdziuk, "An Automatically Generated Evaluation Function in General Game Playing," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 6, no. 3, pp. 258–270, 2014.

[26] G. M. J. B. Chaslot, M. H. M. Winands, H. J. van den Herik, J. W. H. M. Uiterwijk, and B. Bouzy, "Progressive strategies for monte-carlo tree search," *New Mathematics and Natural Computation*, vol. 4, no. 03, pp. 343–357, 2008.

[27] M. Świechowski and J. Mańdziuk, "Self-Adaptation of Playing Strategies in General Game Playing," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 6, no. 4, pp. 367–381, Dec 2014.

[28] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel *et al.*, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," *arXiv preprint arXiv:1712.01815*, 2017.

[29] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, ser. NIPS'13, 2013, pp. 3111–3119.

[30] Q. Le and T. Mikolov, "Distributed representations of sentences and documents," in *Proceedings of the 31st International Conference on Machine Learning - Volume 32*, ser. ICML'14, 2014, pp. II–1188–II–1196.

[31] M. E. Glickman, "The glicko system," *Boston University*, 1995.

Wall Building in the Game of StarCraft with Terrain Considerations

Martin L. M. Rooijackers and Mark H. M. Winands
Games & AI Group, Department of Data Science and Knowledge Engineering, Maastricht University
Maastricht, The Netherlands
mlm.rooijackers@student.maastrichtuniversity.nl, m.winands@maastrichtuniversity.nl

***Abstract*—StarCraft is a Real-Time Strategy game, which has a large state-space, is played in real-time, and commonly features two opposing players, capable of acting simultaneously. One of the aspects of the game is building walls. In this paper, we present an algorithm that can be used for wall building for an agent playing the game of StarCraft: Brood War.**

***Index Terms*—StarCraft, Wall building, Real-Time Strategy game**

I. Introduction

One of the aspects of Real-Time Strategy (RTS) games is wall building. The general strategy of wall building is placing down structures such that you are safe from an attack at a given position. Usually this is done to protect your base and production facilities without having to rely solely on combat units.

This topic is not unique to StarCraft. Previous work has been done in the game Empire Earth [1], where the developers used Graham Scan to decide where to place a wall. A generic wall-building algorithm was also presented in [2].

In StarCraft: Brood War, wall building requires an extra property, which these algorithms do not take into account. Namely, buildings have gaps between them (see Figure 2). Depending on the size of the unit, walls might not be closed off for all units. This sometimes is desirable as well, in case you want your own (small) units to pass through, but do not want the enemy units to pass your wall.

For wall building specifically for StarCraft, there has been an approach that uses answer set programming to add this extra constraint [3]. By specifying the gaps and other constraints through a declarative language, Certicky was able to use the ASP solver clingo to create wall while considering building gaps. This approach has been improved by Richoux *et al.* [4].

The problem with these approaches is that none of them address the gap between buildings and the terrain (mainly cliffs and other natural obstacles). Because of this, some of the walls created with these methods will still have gaps in them that let enemy units through. We propose an algorithm based on the pathfinding algorithm A^* to ensure that the wall from our algorithm will stop whichever enemy unit it is supposed to stop. The advantage of this method is that the constraints are checked by checking if a path exists in the game itself, thus ensuring that a wall found by our algorithm is tight. With an extra calculation step, our algorithm can also create walls where smaller units can pass through, but the larger units of the opponent cannot.

This paper is structured as follows. First, Section II gives the problem definition of wall building. Next, Sections III and IV discuss the wall-building algorithm. Section V describes the pseudocode implementation of the wall-building algorithm. Section VI describes the experimental setup and the results of the experiments. Finally, Section VII draws conclusions from the results and presents future work.

II. Problem Definition

In the RTS game of StarCraft: Brood War, a wall is a set of structures and units placed in such a way that no enemy unit can pass from one side of the wall to the other. In this section we describe the problem that our algorithm tries to solve.

The StarCraft map consists of two grid types: the walk grid, where each cell is an 8×8 pixels square, and the build grid, where each cell is a 4×4 walk tile square (hence 32×32 square of pixels). Some of these build tiles are buildable, and others are not due to natural obstacles (e.g., cliffs) or due to the game rules (buildings cannot overlap).

Each building has a build size and a real size. The build size indicates the height and width of the building in terms of building tiles. For example, a Terran supply depot has a build width of 3 and a build height of 2. The real size indicates how much walkable space the building takes up. In the case of StarCraft: Brood War, the actual space taken up by some of the buildings is less than its build size $\times$ 32 pixels, leaving some additional space for passing it. This causes gaps to be created between buildings placed next to each other and to natural obstacles (cliffs have these gaps as well).

Thus the wall-building problem is about finding locations for buildings such that a given enemy unit (given in pixel width and height) cannot pass through the wall. An extra constraint can be that another smaller unit has to be able to pass through while the given enemy unit still cannot pass through (this allow for hit & run tactics where ranged units can retreat behind a wall after shooting).

The algorithm presented in this paper is specifically for checking if a given set of buildings can satisfy these constraints. The problem of determining the minimum number of buildings required for building a wall is beyond the scope of this paper. This however can be easily added by having an algorithm generating a list of buildings and using the algorithm

978-1-5386-4359-4/18/$31.00 ©2018 IEEE

presented in this paper to check if those buildings can form a wall, until a suitable combination of buildings is found.

III. Approach

Fig. 1. Example of a wall found using the wall-building algorithm. Choke point is indicated with a red line and a purple circle.

In our approach we use a wall to seal off a choke point. A *choke point* is a location on the map (terrain) that connects two regions [5] (see also Figure 1). If a wall is built near this choke point, then the two regions of the choke point are separated (no longer reachable by ground units).

Since both buildings and units can be used in a wall, we use the term structure to indicate a part of a wall (either a building or a unit). The first part of the wall-building algorithm requires a way to generate a set of structure locations, which can potentially form a wall. In the game of StarCraft, units and buildings cannot overlap, buildings cannot overlap with unsuitable ground and units cannot overlap with unwalkable ground. We use a depth-first search algorithm to determine possible structure locations. At the initial depth, the algorithm places a structure close to the choke point. After selecting a location for the first structure, the next depth includes a new structure that is placed adjacent to any of the structures already placed. This is done because a wall should not contain a gap. Therefore each building and unit is adjacent to at least one other building or unit. This process continues until all structures have been placed.

Each time that the placement algorithm placed all structures (reached maximum depth), the validation algorithm checks if the placement of the structures forms a wall. Determining if a set of structures forms a wall requires checking if there is a path that goes from one side of the choke point to the other side that passes through the choke point that is supposed to be walled of. For this we compare two different methods. In both methods we limit the search to a 16×16 build grid around the choke point to ensure that all possible paths have to go through the choke point that is supposed to be walled of. The first method that we use is a flood fill starting from one side of the choke point. If the flood fill reaches the other side of the choke point then the algorithm will generate the next set of possible structure locations which can potentially form a wall. The second method is the A^* algorithm. The A^* algorithm tries to reach the other side of the wall based on heuristic search instead of a brute force approach like flood fill.

IV. Wall-Building Algorithm

The wall-building algorithm used to calculate a wall is based on a depth-first search approach to find possible structure placements combined with flood fill or A^* to check if a structure placement is a wall. Just like the declarative programming approach from Certicky [3], we construct a 16×16 grid around a choke point (see Section III for the definition of a choke point). Besides the choke point location, the algorithm also requires the list of buildings and units available to create the wall.

The wall placement algorithm starts off with no structures placed (depth-first search at a depth of 0). At this point (depth) the algorithm will pick one of the possible structures and place it down at a location close to the choke point. Determining a possible location is performed by checking if every tile occupied is buildable (if it is a building) or walkable (if it is a unit). After the initial placement, the wall placement algorithm places the next structure adjacent to the initial structure. This adjacency is 8-ways (horizontal, vertical, and diagonal). Every next structure from this point is then placed adjacent to at least one of the structures already placed. This process continues until all available structures have been placed.

Once all structures are placed (at the leaf node), the algorithm starts a flood fill or A^* from a tile on one side of the wall. Both the flood fill and A^* algorithm try to reach a tile on the other side of the choke point by passing through the choke point. If successful, the current placement of structures is not a wall. In this case the depth-first search backtracks and tries a different placement of structures. Besides walkable gaps on the tiles not covered by the structure, the flood fill and A^* can also pass between gaps formed by buildings (see also Figure 2). Each building has a certain number of pixels as a gap on each side (top, bottom, left, and right). When a side of a building is adjacent to the side of another building, the corresponding sides combine the pixel values.

The algorithm used by Certicky *et al.* [3] did not take into account that two buildings placed diagonally also create a gap. Thus this algorithm sometimes misses a possible wall, because it does not take the gap into account (see also http://wiki.teamliquid.net/starcraft/Walling).

The algorithm from Richoux *et al.* [4] does not take the terrain (cliffs and other natural obstacles) into account. These terrain features have gap values as well. Although the gap values of buildings are known, it is not yet known what the exact gap values of all terrain features are. Therefore, our algorithm uses the walk grid to calculate if a unit can fit through the gap of a building and a cliff or other terrain feature. Even though the gap size is not known, the Brood War API

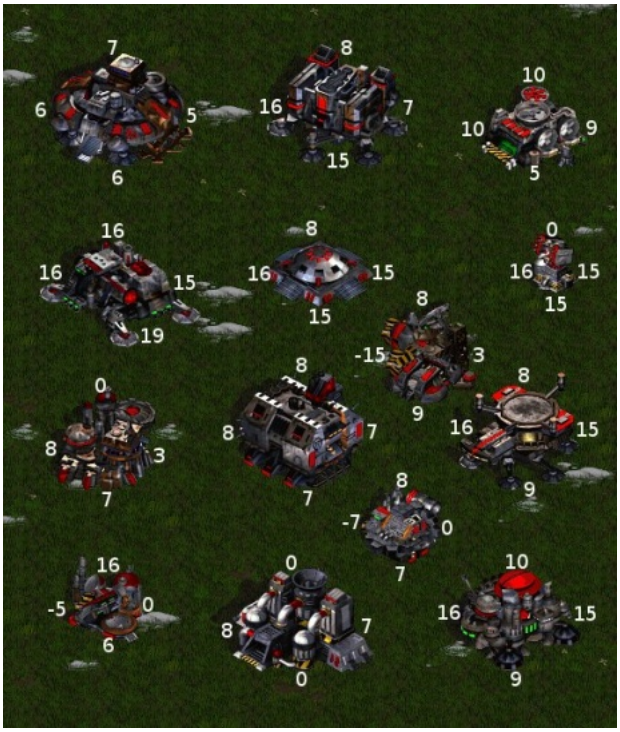


Fig. 2. Gaps between Terran buildings in pixels. The numbers indicate the size of the gaps in pixels. Picture taken from teamliquid wiki.

does give information about which tile can be walked on. Thus instead of calculating the gap between a building and a cliff, we instead take the total number of walkable tiles between a building and a cliff. This is then multiplied by the pixel size of a walkable tile (8×8, see Section II). This estimation is an upper bound to the gap size, so the walls created with this estimation will stop the unit it is supposed to stop. However, if the actual gap is smaller than the estimation, our algorithm might miss a possible wall. Determining the exact gap size when terrain is involved is left for future work.

The exact details of this process is an implementation detail. The code for the wall building can be found at https://github.com/MartinRooijackers/LetaBot.

V. Implementation

The wall-building algorithm requires a choke point, a tile on one side of the choke, a tile at the other side of the choke and a list of structures. The other information about buildable and walkable locations can be derived from the BWAPI. This pseudocode gives a high level overview of the implementation for a StarCraft agent.

The wall-building algorithm requires the following input:

- *S*: a list of structures, each containing its width and height
- *B*: a list of structures already placed, each containing its width and height and (x, y) position
- *choke*: tile location of a choke point
- *sTile*: start location of the flood fill
- *eTile*: end location of the flood fill
- *Enemy*: (width, height) tuple indicating the size of the unit that should not be able to pass through the wall

The algorithm is split into two components: the wall placement algorithm that determines valid structure locations and a wall validation algorithm that checks if these locations form a wall. The second step can be performed by either A^* or flood fill. The wall placement algorithm is given in the pseudocode below (see Algorithm 1). The algorithm starts with calling the WallIn function with the parameters mentioned above. This function calls StructurePlacement, which uses depth-first search to place the structures. As long as a structure needs to be placed, the algorithm uses the ValidLoc function to determine a place to put the building or the unit. This process is repeated until all structures are placed. Once all structures are placed, the algorithm checks whether the structure placement is a wall with the CheckWall function. Two variants of this function have been implemented by either using a flood fill or A^*.

Algorithm 1 WallPlacement

procedure WallIn($S, choke, sTile, eTile$)
  StructurePlacement($0, S, \emptyset, choke, sTile, eTile, Enemy$)
end procedure

procedure StructurePlacement($depth, S, B, choke, sTile, eTile, Enemy$)
  if size(S) $= 0$ **then** ▷ all structures placed
    for all $x \in \{choke_x - 8, \ldots, choke_x + 7\}$ **do**
      for all $y \in \{choke_y - 8, \ldots, choke_y + 7\}$ **do**
        Visited$(x, y) \leftarrow$ False ▷ clear last flood fill
      end for
    end for
    isWall $\leftarrow$ CheckWall($sTile, eTile, Visited, Enemy$)
    if $isWall = false$ **then**
      return ▷ This is not a wall in, generate a new structure location
    end if
    if $isWall = true$ **then return**
      Output/Store current structure locations and end algorithm
    end if
  end if
  for all $x \in \{choke_x - 8, \ldots, choke_x + 7\}$ **do**
    for all $y \in \{choke_y - 8, \ldots, choke_y + 7\}$ **do**
      if ValidLoc$(x, y, S_0, depth)$ **then**
        StructurePlacement($depth + 1, S \setminus S_0, B \cup \{(S_0, x, y)\}, choke, sTile, eTile, Enemy$)
      end if
    end for
  end for
end procedure

The ValidLoc function (see Algorithm 2) checks whether a building can be placed at a certain location. Since a wall requires all buildings to be adjacent, this function also checks whether the build location is adjacent to another location

already occupied. The only exception is the first building, since it cannot be adjacent to anything yet.

Algorithm 2 ValidLoc

```
1: procedure VALIDLOC(x, y, struct, depth)
2:     Adjacent ← False
3:     for all xTile ∈ {x, ..., x + (struct_w − 1)} do
4:         for all yTile ∈ {y, ..., y + (struct_h − 1)} do
5:             if Occupied(x, y) = True then ▷ BWAPI function
6:                 return False
7:             end if
8:             if Tile adjacent to other structure then ▷ 8-way
9:                 Adjacent ← True
10:            end if
11:        end for
12:    end for
13:    if Adjacent = False ∧ depth ≠ 0 then ▷ adjacency check
14:        return False
15:    end if
16:    return True
17: end procedure
```

There are two ways to implement the CHECKWALL function used in the wall placement algorithm. The first option is to use the flood-fill algorithm. The second option is to use the A^* algorithm. Both algorithms use information from the BWAPI to determine if a position is invalid. If a position is invalid, it cannot be traversed. An invalid position is a position where:

- The *x* or *y* position is outside of the map.
- The *x* or *y* position is outside of the 16×16 grid.
- The gap between buildings is not large enough (see Figure 2).

Hence, a valid position is a position on the map that does not have these characteristics. The flood-fill algorithm uses 8 directional movements. For the implementation of A^*, we use the Manhattan distance heuristic. Since the standard variant of the algorithms are used, the pseudocode is not reproduced here. The implementation details can be found in the source code.

VI. EXPERIMENTS

A. Setup

In the first experiment of this paper, we investigated the computing time of the wall-building algorithm. For this, we have used the standard "1 barracks + 2 supply depots" to wall off the starting location. This configuration is the standard build order that is used in professional games where the Terran player wants to protect the starting location from rush strategies. We picked four CIG maps from the 2017 tournament and one from the general CIG map pool. The maps we selected are:

- Hitchhiker 1.1
- Tau Cross 1.1
- Neo Aztec 2.1
- Andromeda 1.0
- Python 1.3

We have run the test on each map 20 times. The mean running time of the flood fill and A^* variant of the algorithm can be found in Table I. The table reveals that A^* decreases the computation time considerably. The standard deviation can be found in Table II. It shows that if our algorithm finds a wall, it will do so quickly. However, if a wall does not exist, our algorithm will try all possibilities, which causes the high deviation.

Map/Algorithm	Flood Fill	A^*
Hitchhiker 1.1	11.6s	6.5s
Tau Cross 1.1	2.6s	1.7s
Neo Aztec 2.1	0.6s	0.7s
Andromeda 1.0	4.3s	3.3s
Python 1.3	15.3s	11.4s

TABLE I
RUNNING TIME OF EACH ALGORITHM VARIANT IN SECONDS. AVERAGE OF 20 EXPERIMENT RUNS.

Map/Algorithm	Flood Fill	A^*
Hitchhiker 1.1	11.0s	5.8s
Tau Cross 1.1	1.3s	0.6s
Neo Aztec 2.1	0.07s	0.7s
Andromeda 1.0	2.6s	0.2s
Python 1.3	14.4s	9.5s

TABLE II
STANDARD DEVIATION FROM THE RUNNING TIME OF EACH ALGORITHM VARIANT IN SECONDS.

We have used this algorithm to give our StarCraft agent LETABOT the capability to build a wall in order to stop a rush build. Such rush builds are used by professional StarCraft player and bots. Our wall-building algorithm was first used in the CIG 2014 tournament. It has been used in every major StarCraft AI tournament ever since. The effect of the wall placement is especially notable when our agent plays against a rush build, which is a popular strategy in the StarCraft AI tournament. We got the following notable tournament results with the help from this wall-building algorithm:

- CIG 2014: 3rd place
- CIG 2016: 3rd place
- CIG 2017: 4th place
- AIIDE 2014: 3rd place
- AIIDE 2016: 4th place
- SSCAI 2014: 1st place mixed+student
- SSCAI 2015: 1st place student division
- SSCAI 2016: 1st place mixed+student
- SSCAI 2017: 2nd place student division

In the last series of experiments we also tested what would happen if LETABOT did not use the wall-building algorithm. For this we disabled the wall-building algorithm in LETABOT and put it up against two rush bots (CARSTEN NIELSEN and

WULIBOT) on the 5 maps from the first experiment. The results of that can be seen in Table III, winning only 51% ($\pm 9.8\%$) of the games. If the wall-building algorithm is turned on, our bot scores a 100% win rate against these bots.

What is noticeable, is that the wall-building algorithm does not add much when playing on a large map like "Andromeda", where the size alone makes rush strategies less effective. Maps where you start on the high ground and have a ramp that can be used as a choke point, help in the defensive without walls as well ("Hitchhiker" and "Python"). But with the exception of large maps like "Andromeda", LETABOT benefits from using a wall-building algorithm to deter rush strategies.

Map/Bot	CARSTEN NIELSEN	WULIBOT
Hitchhiker 1.1	4-6	3-7
Tau Cross 1.1	5-5	0-10
Neo Aztec 2.1	10-0	0-10
Andromeda 1.0	10-0	10-0
Python 1.3	7-3	2-8
Total	36-14	15-35

TABLE III
WIN RATE OF LETABOT WITHOUT USING A WALL-BUILDING ALGORITHM (FORMAT: WINS-LOSES).

VII. CONCLUSIONS & FUTURE RESEARCH

In this paper, we have shown two variants of an algorithm that can be used for building walls in StarCraft. Unlike other methods, this algorithm guarantees that a wall can be used to ensure that a given unit cannot pass through it. The downside compared to other methods is that iterating through the possibilities to ensure that the wall is tight, is a costly calculation. The A^* heuristic search improves this, but this algorithm is still mainly recommended to be used for pre-calculating building positions to ensure a tight wall. Because the map of StarCraft is static, this information can be calculated and stored, such that it can be retrieved next game and be used immediately. Thus this algorithm becomes a tool, like the terrain analysis tool BWTA, which is used by our StarCraft agent for choke point analysis and splitting the map in regions.

One of the things still missing from the pathfinding is the exact data on gaps created by the terrain. For now our algorithm used the walkable data given by the BWTA. The walls created by this are tight, but the criteria are stricter than they have to be. Thus our algorithm sometimes report that there is no wall possible, even though one exists. This explain the large variance of running time between maps, since our algorithm takes less time if it finds a wall (because then it can terminate the search). Most of the time, an alternative wall (further away from the starting position) will be found at the cost of extra running time. This is especially the case on maps like "Python".

A way to improve the running time is to have some extra checks in place for the structure placement to reduce the number of placement choices that can be trivially calculated not to be walls.

REFERENCES

[1] T. Teich and I. Davis, "AI Wall Building in Empire Earth II," in *AIIDE*, 2006, pp. 133–135.

[2] M. Grimani, "Wall building for RTS games," *AI Game Programming Wisdom*, vol. 2, pp. 425–437, 2004.

[3] M. Certicky, "Implementing a wall-in building placement in StarCraft with declarative programming," *arXiv preprint arXiv:1306.4460*, 2013.

[4] F. Richoux, A. Uriarte, and S. Ontañón, "Walling in strategy games via constraint optimization." in *AIIDE*, 2014, pp. 52–58.

[5] L. Perkins, "Terrain Analysis in Real-Time Strategy Games: An Integrated Approach to Choke Point Detection and Region Decomposition," in *Proceedings of the Sixth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE 2010*, G. M. Youngblood and V. Bulitko, Eds. The AAAI Press, 2010, pp. 168–173.

Explainable AI for Designers: A Human-Centered Perspective on Mixed-Initiative Co-Creation

Jichen Zhu
Drexel University
Philadelphia, USA
jichen.zhu@drexel.edu

Antonios Liapis
University of Malta
Msida, Malta
antonios.liapis@um.edu.mt

Sebastian Risi
IT University of Copenhagen
Copenhagen, Denmark
sebr@itu.dk

Rafael Bidarra
Delft University of Technology
Delft, The Netherlands
R.Bidarra@tudelft.nl

G. Michael Youngblood
Palo Alto Research Center
California, USA
Michael.Youngblood@parc.com

***Abstract*—Growing interest in eXplainable Artificial Intelligence (XAI) aims to make AI and machine learning more understandable to human users. However, most existing work focuses on new algorithms, and not on usability, practical interpretability and efficacy on real users. In this vision paper, we propose a new research area of eXplainable AI for Designers (XAID), specifically for game designers. By focusing on a specific user group, their needs and tasks, we propose a human-centered approach for facilitating game designers to co-create with AI/ML techniques through XAID. We illustrate our initial XAID framework through three use cases, which require an understanding both of the innate properties of the AI techniques and users' needs, and we identify key open challenges.**

***Index Terms*—explainable artificial intelligence, mixed-initiative co-creation, human-computer interaction, machine learning, game design**

I. Introduction

With the swift development of artificial intelligence (AI) and machine learning (ML) in recent years, their applications (digital games included) have become more sophisticated. With the rise of algorithmic complexity, however, it is becoming increasingly difficult for humans to understand these algorithms and hence to have trust in them. For instance, while recent development of deep learning techniques produced impressive results, it is notoriously difficult for humans (programmers included) to gain full insights into the system's function.

In this vision paper, we focus on one group of human users. We propose a new research area of *eXplainable AI for Designers (XAID)* and specifically for game designers. The increase in game AI sophistication opens up a new creative design space for potentially new gameplay and/or more efficient production. However, game designers (such as rule designers, level designers and artists) often find these techniques inaccessible and difficult to explore their full creative potentials without a deep understanding of how they function. To the best of our knowledge, there has been a lack of XAI research to address this particular problem.

By focusing on a specific user group, their needs and tasks, we provide the basis of a human-centered XAID approach which facilitates game designers to co-create with AI/ML techniques. XAID can enhance game designers' capabilities to co-create playable experiences with AI, including but not limited to ML, agent control, procedural content generation, and planning. We believe that, although fundamental understandings of the properties of different AI/ML techniques are essential, the goal of XAID includes investigating the actual *usability* of XAI in terms of how it supports game designers in specific design tasks.

Below, Section II presents related work on XAI and mixed-initiative human-AI co-creativity. We present our framework on explainability and the three axes of XAID in Sections III and IV. Through three use cases, we illustrate our initial framework for XAID, requiring understanding both the innate properties of the AI/ML techniques and users needs. Finally, we identify key open challenges for future XAID research.

II. Related Work

Current XAI research can be classified by the types of techniques being illuminated (e.g. black-box techniques, white-box techniques). Given the limited research on XAI for game design, we also provide background on mixed-initiative co-creation systems where AI and human designers work together, an interaction model we envision XAID to extend. Finally, we review current evaluation methods of XAI.

A. Black-Box XAI approaches

Current XAI approaches for black-box systems such as neural networks can be roughly divided into approaches that aim to (a) visualize features, and (b) elucidate the relationship between neurons. Visualizing hidden layers' features [1]–[4] can give insights into what network inputs would cause a certain reaction in an internal neuron or in the network output. In contrast to optimizing the network's weights, as normally done through gradient to train the network, a researcher can use the same process to optimize an image that would maximally activate a certain neuron. These techniques can help to identify what certain neurons in a DNN pay attention to.

While these feature visualization techniques can offer insights into particular neurons, other approaches aim at under-

978-1-5386-4359-4/18/$31.00 ©2018 IEEE

standing how multiple neurons in a network interact to reach a decision. Techniques that aim to explain relationships between neurons are known as attribution and a variety of different approaches exists [2], [5], [6]. One of the simplest attribution examples is the saliency map, which is a heatmap highlighting which areas of the input image are most responsible for reaching a certain output classification.

These approaches—especially when combined—offer some insight into the inner workings of a neural network, making DNNs more of a grey than a black box. While earlier work tried to address this problem by e.g. creating decision rules or a decision tree out of a neural network [7], how these approaches will scale to modern DNNs is an open problem.

In a recent article [8], Olah *et al.* demonstrate how these interpretability building blocks can be combined in a unified interface to gain a deeper understanding of the workings of a neural network. We believe these techniques could help in creating compelling interfaces for designers.

B. White-Box XAI approaches

There is a large body of work in helping human users better understand white-box AI techniques, whose inner workings are transparent (e.g., simple decision trees). Earlier work of XAI can be traced to expert systems and Bayesian networks [9]. In a review, Lacave and Díez [10] categorized existing approaches into three main types: explanation of evidence, explanation of the model, and explanation of the reasoning. Notably, they pointed out a serious limitation in research which focused mainly on theoretical models of explanation without empirically validating these approaches with human users. We argue that the current state of XAI, including both white- and black-box AI, shares a similar limitation.

In the domain of planning, research in *plan explanations* attempts to make the systems' output more understandable through a more understandable representation of plans [11] and by generating explanations using natural language [12]. More recently, the focus has shifted from explaining the plans themselves to explaining how the planner produces its output [13]. There is growing interest in explaining the planner's behavior through verbalization [12], [14].

Specifically for games, limited work exists on explaining the underlying white-box systems. As an example, the graphical representation of behavior trees has made it easier for game designers and artists to understand how the underlying AI functions. Another loosely related work is on explaining utility AI: [15] annotates positions in a 3D shooter game based on their strategic value (e.g. at the right distance to an enemy with coverage from secondary threat). However, there is very little work on explaining white-box AI techniques for the purpose of facilitating design tasks.

C. Mixed-Initiative Co-Creative Systems

We envision XAID as a useful way to facilitate game designers in their work. Interfaces intended to help designers create content and, more broadly, design games have long been challenged to provide appropriate, informative feedback to their end-users. Game engines and their editors offer a variety of intuitive interfaces for simplifying a user's tasks. Through the use of AI, these computer-aided design tools are elevated to *mixed-initiative co-creative systems* [16] where 'both the human and the computer proactively make contributions to the problem solution, although the two initiatives do not need to contribute to the same degree'. Likening the design process to a dialog between colleagues [17], computational initiative can refer to the task initiative (i.e. who initiates the dialog), speaker initiative (i.e. when each actor will speak, and whether actors can interrupt each other), and outcome initiative (i.e. who decides when the dialog is finished or the problem is solved). The dialog analogy clarifies how explainable AI is vital in conveying to the user its reasons in taking any of task, speaker or outcome initiatives mentioned.

Numerous mixed-initiative co-creative tools have been developed over the last decade for game design, although for the most part as academic rather than commercial endeavors. Many of these tools focused on explaining the properties of game design artifacts that the computational designer produced for direct use or for further editing by the human designer [18]–[20]. For example, *Sentient Sketchbook* [18] autonomously creates levels as alternatives to what the designer is currently doing, and there is no explanation regarding such a task initiative. If the designer stops and observes each computational suggestion, the interface displays numerically which functional level properties (e.g. area balance or exploration) improve or decrease compared to the current human sketch. Through fairly simple visual feedback (e.g. plus and minus signs), the tool attempts to explain why this suggestion could be desirable or undesirable to the designer.

Numerous mixed-initiative tools have focused on visualizing such properties of their specific artifacts for each user (and in the case of Danesh [20], properties of a large sample of artifacts). However, there is little research in explaining the creative process (rather than the final artifact) in co-creative tools, and all attempts to date have focused on visualizations rather than on natural language generation of the explanation. On the other hand, there have been several interesting attempts at explaining autonomously creative processes (without a designer involved either as a co-creator or as a consumer of the explanation) both in game generators such as *Angelina* [21] and in broader creative software such as *The Painting Fool* [22]. Many of the positions in this article, especially in Section VI-A, borrow from these white-box generative systems.

D. Measuring Explanations

While there is no established definition of explainability and how to measure it, ultimately explanations serve to build understanding and possibly trust between the AI and the user or beneficiary of the AI. Testing understanding of software has a long history in human computer interaction and education. For complex mechanisms of AI and ML, understanding requires testing model induction: how does the induced mental model a person holds match or differ from the actual model?

The typical route for model testing is through instance testing: given a specific instance, can a human predict or determine how the model will act? Complex instances may be decomposed into sub-instances or competencies and tested in smaller measures to determine the level of model match and understanding. The learning rate, precision, and recall of the process and the induced model are important factors. Explanations that improve these can lead to a qualitative measure for comparison between explanations. Some explanations may require more mental processing to learn, so task loading is also a consideration. Measuring trust in machines is a complex issue and still an active area of research in psychology [23].

III. Explaining Explainability

Developing XAID to facilitate design tasks first requires a thorough understanding of explainability and how it connects to the properties of different AI/ML techniques. Interest has been increasing in better understanding some of the learned AI models, specifically in the field of machine learning. Techniques that are reduced into networked structures of weights with complex topologies and varying transformation functions embedded in them are notoriously difficult to understand by humans. Yet, with the recent advances in deep learning, these models are being used in a broadening and critical set of everyday life applications. Reliance on ML for critical tasks and especially those involving human-life requires trust, which is typically gained through some level of transparency that facilitates a comfortable level of model understanding for the person giving that trust [24].

Explainability is not just needed in opaque, machine-learned models, but in many facets of AI. We define explainability as being clear of obscurity and understandable in all aspects. This means that to truly understand something, we must be able to introspect all of its mechanisms. Some argue that this a white-box view of explainability; we maintain that this is the only true explainability: the ability to answer *why* questions.

Axiom 1: **Explanation without introspection is not explanation.**

An argument can be made that some reactive (black-box) techniques are fully understandable from observation of all potential combinations of input and their related outputs. This black-box view does provide an understanding of behaviors, but does not address the obscurity of the underlying model, and thus we call this Observable AI, which is valuable and may suffice for proper model induction in humans. Observation, however, is not an explanation: it cannot truly answer *why* it does what it does. It is also important to note that black-box testing of complex models may be intractable, so observable behavior may have a level of uncertainty that matches the inconsistencies and incompleteness of the observations made.

Axiom 2: **Understanding through external probing is observation.**

If we look at the spectrum of AI techniques as shown in Fig. 1, we can reduce them to a dimension of reactive to deliberative—or through Daniel Kahneman's lens [25], fast and slow thinking[1]. Performing thinking as a reflex, which is a stimulus-driven control policy, is something that an agent may have been created with. Such a thought (which came into existence as a black box) can not be inspected and may never be truly explainable.

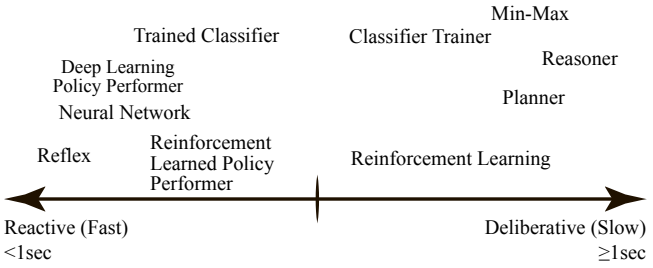


Fig. 1. AI/ML Techniques on a reactive to deliberative Scale.

Axiom 3: **Reactive elements that have always been reactive from inception are not explainable, but may be observable.**

Many reactive elements are created through deliberative training processes as shown in Fig. 2. Deliberative processes have the property of all being inspectable and procedural. This is not to say that all deliberative techniques are explainable, but their obscurity comes from complexity (e.g., processes with many steps, stages, or interoperable rules), not reduction.

Conjecture 1: **Deliberative elements may require explainability due to complexity.**

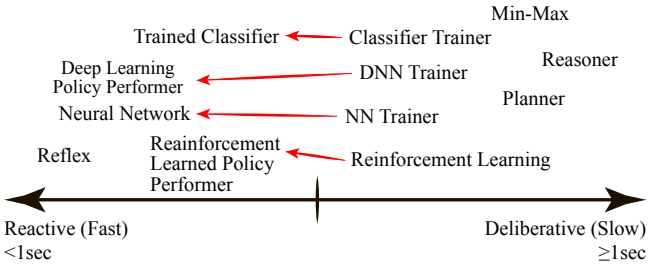


Fig. 2. Mapping deliberative AI/ML techniques to reactive processes.

Obscurity in reactive elements comes from dimensionality reduction of the state space or the data. This reduction is the condensation of training data into a model, by transforming data and state space topology from the deliberative process to the reactive.

Conjecture 2: **Reactive elements may require explainability due to reduction of training into a model through data and topology transforms.**

If we want to understand and inspect a reactive model, the explanation lies in the deliberative portion that created the model. This typically has a temporal aspect to it. The

[1]For a complex agent, such as humans, the line between how much of what an agent uses for decision-making exists in the reactive versus the deliberative scale is an open area of research.

exception may be one-shot learning techniques, which is a topic for another paper.

Axiom 4: **The explanation of a reactive model lies in the deliberative process that created it.**

For reactive processes, the ways in which eXplainable AI (XAI) techniques may manifest themselves is via injection of explainable notes to the reactive process; these notes are created by the deliberative process(es) that led to the reactive process, and provide a surface for post-hoc analysis of the training of the reactive model in the deliberative process(es). Alternatively, the deliberative process(es) may generate a separate explanation in the process of creating the reactive model. These explanations serve to illuminate the reduction process and the impact of what is learned that drives the reactive model behavior. These aim to largely answer the *what* and *why* details of decisions the model will make based on perception or who, what, when, and where influenced the behavioral response. It may also answer the more mechanistic *how*.

For deliberative processes (such as those shown in Fig. 2), the ways in which XAI techniques may manifest themselves is by making the complex tractable for human understanding. It is a reduction, reorganization, or reframing of the complex into something understandable that maintains the transparency and introspection of the model.

IV. eXplainable AI for Designers (XAID)

It is generally agreed upon that the goal of XAI is to increase users' trust, their ability to interact with the systems and with their decisions, and improve the transparency of the system [13]. However, most existing work focuses on new algorithms of XAI rather than on usability, practical interpretability and efficacy on real users [26]–[28]. Although we believe that fundamental understandings of the properties of different AI algorithms are an essential part, XAI techniques should be developed with specific users and their needs in mind if they are to fulfill their promise.

We propose a new area of research in eXplainable AI for Designers (XAID) who create interactive digital products built on AI components. As AI and ML techniques are mature enough to reach commercial products (e.g. computer games, virtual assistants, smart objects), designers need to understand how the AI component works in order to devise desirable ways for the end-users to interact with the systems. Unlike the end-users of an AI system, designers constitute a unique user group because they not only consume the results of AI systems, but also *co-create* with them. To the best of our knowledge, no XAI work focuses on designers and co-creation.

In the rest of this paper, we focus specifically on game designers (such as rule designers, level designers and artists) who do not have a strong technical background in AI. The key purpose for XAID is to (a) provide designers with sufficient understanding of the underlying AI system and its behavior, and hence (b) facilitate their design tasks through co-creation. Our positions are the following:

1) Work in XAID needs to build on understanding of the nature of underlying AI techniques. As argued above, different AI techniques afford explanations with introspection while others afford only observations. Although both can be useful for designers, understanding the option of explanation and/or observation can help shape how XAID can support the co-creative process between designers and AI.
2) Work in XAID needs to center on specific human users (e.g. game designers) and their specific needs and tasks. Compared to more general XAI research, XAID as proposed here has the advantage of a more concrete if narrow scope. Through the three specific use cases in Section VI, we argue that work in XAID should be designed for and evaluated with specific users.

V. Mapping the Space of XAID

In a broad stroke, we describe the XAID space along three main axes, each one spanning its own spectrum.

A. Spectrum of Explainability

We first identify the XAID spectrum of explainability, ranging from *explanations* that provide introspection into the operation of AI techniques to *observations* that offer insights of the input-output pattern. Explanations can provide designers with information such as the chain of actions and why the algorithm takes a specific decision. Observations, for instance, can be used to inform game designers of all the possible actions an AI-controlled non-player character (NPC) will take at a given game state and the likelihood of each action.

Although the decision between offering explanations or observations relate to the properties of the underlying algorithms, as argued above, it also depends on the needs of the designers and their tasks. For example, observations may be the best choice for level designers working with a white-box NPC AI because understanding the exact operation of the AI is not necessary to their design task and may cause information overload. By contrast, a game designer tasked with game balancing in the same project may need to know exactly why the NPC performs certain actions—especially if they are unexpected—in case the game attributes themselves (which may influence these decisions) need to be corrected.

B. Spectrum of Initiative

A crucial aspect of any AI-assisted design system is the degree and type of initiative that it can take in performing its tasks. The spectrum of initiative traces the limits of a system's intervention, and it determines the kind of explanations that it may be expected to provide. We can distinguish three bands in this spectrum, corresponding to the *level of system initiative*, each with its typical kind of explanation.

At the lowest level, the system passively waits for the designer to request assistance, e.g. some on-demand analysis. For this, a typical explanation can include a simple description of the task performed, possibly with a number of meaningful parameters used to yield its output.

At the next level, we can devise AI assistance that requires a higher degree of autonomy. Correspondingly, the kind of explanations involved in their execution has an increasing complexity. Some examples of these (together with a possible explanation) are:

- explore how to proceed (e.g. describe the space of possible alternatives, sampling methods used, evaluation criteria);
- sketch a range of choices (e.g. characterize the extreme points of a range of options, to give insight into what it involves);
- warn a designer regarding some risk ahead (e.g. look ahead for what-if analysis, to identify and describe conflicts or risks)

The explanations mentioned in these examples require a considerable understanding of the processes and goals at hand.

Going even higher on the spectrum of autonomy and initiative, we can devise an AI system working on par with the human designer, taking on activities more as a colleague than as an AI assistant. At this level the tasks, outcomes and explanations are currently only expected from a human co-designer, not an AI assistant. Examples of these could be:

- making informed design choices (e.g. based on the awareness of the goals of the design task);
- intervene to suggest the best way to proceed, e.g. switch focus to another task, attempt some alternative solution or try to avoid an early commitment (justifying such suggestions requires a much deeper understanding of both the design situation, its history and the available options);
- signal and correct a 'mistake' made by the designer (this requires explaining why it is perceived as a design mistake, e.g. violating some previously stated intent, and finding out alternatives with a better outcome);
- propose a sensible task division (explaining such a proposal will likely need a formidable amount of knowledge; in addition to the above, this typically human activity requires meta-knowledge on both the nature of each subproblem, their relation to the ultimate goal sought and the competences of team members, AI or otherwise).

For some more down-to-earth activities, one can imagine them taking place at any level of the spectrum above. For example, the creation of a specific type of content could be either explicitly issued from a procedural content generation (PCG) algorithm by a designer, suggested by the system at some appropriate stage, or autonomously performed as a fitting complement to whatever else the designer is doing. However, explanations on that same activity will likely have to vary according to the level of initiative being taken.

C. Spectrum of Domain Overlap

Another aspect of XAID is the amount of overlap between the tasks performed by the designer and the tasks performed by the AI. To a certain degree, this can be considered the degree of co-creativity that is needed, and can similarly affect how (or how much) each task by the AI needs to be explained. The spectrum of domain overlap ranges from the scenario that AI and a human designer making use of the same tools applied to the same task (*on-task* co-creative activities) to the scenario that a human designer is working on an aspect of the game while the AI handles another that only tangentially be affected by the designer's input (*off-task* co-creativity).

Let us consider how the explanations differ between on-task and off-task AI co-creativity. If an AI and a designer work on the same domain, e.g. changing the same game level using the same tile-based structure as is the case in *Sentient Sketchbook*, the explanations provided by the AI should be fairly specific as the designer (a) is very aware of the terminology and current problems of the work in progress, (b) can directly observe the elements that the AI refers to, and (c) must be able to take immediate decisions regarding the AI suggestions, e.g. to accept or reject them. In an example of an off-task collaborator, we consider a designer who is creating a level which an AI agent playtester attempts to solve, similarly to *Roppossum* [29]. In such a case, the human-made artifact (level) directly affects the AI agent, but the explanations provided by the playtester should focus on level-specific concerns (e.g. "this platform is 90% likely to cause me to overshoot, if the player has poor reflex time"), rather than explanations of its behavior. The level designer may not be knowledgeable of (or interested in) the AI agent's internal decision-making priorities, but instead may be interested why the level is deemed unplayable by the AI agent.

VI. Three Use Cases

In order to illustrate the human-centered perspective on XAID described so far, we discuss three different use cases.

A. Use Case 1: White-Box PCG System

The first use case tackles the problem of a level designer who is using a computer-aided design (CAD) tool to create the perfect overworld map for a free-roaming car-racing game similar to *Mad Max* (Warner bros 2015). Apart from typical CAD functionality such as texture brushes, mesh placement and camera movement, the tool can generate the entire level (or parts of the level) on command. The algorithm for this generative component is based on grammars, which have been inserted into the system not by this level designer, but by a tool programmer who may not be working in the same company.

The above instance is one of *on-task* collaboration (the two designers literally work on the same map), and the level of initiative is *on-demand*. Due to the deliberative steps that a grammar generator takes, the algorithm can narrate its generative process as textual output, which is in the form of *explanation*. Given the fact that the generative grammars follow a fairly transparent process, we could conceive that the explanation generator could be included within the procedural generator with a sentence produced after relevant commands, function calls or choices: in this particular case, grammar expansions. This goes beyond a simple log of steps and decisions taken, e.g. it may also carry information on the context influencing those decisions. Generative grammars

have an ideal generative architecture for such an explanation because it is, in many respects, a pipeline: each expansion produces an intermediate output that is passed on as input to a further expansion [30]. Presenting to the designer a compelling and intuitive narrative regarding the choices taken by the PCG system can be done in a variety of ways, including:

- *sequentially* in the order that the system makes decisions. This explanation can follow some form of story structure which simulates e.g. the generative pipeline [22]. In order to enhance (e.g. via natural language processing) how the connections are made between different steps of the generation, we can investigate work on story generation so that the narrative is coherent and causal links are made obvious. This can be achieved, for example, by post-processing the generated sequence of sentences to introduce throwbacks to past generative decisions which affect future outcomes, or to foreshadow how one early decision affects the final outcome.
- *summary of highlights* of the generative process, by filtering out and omitting less interesting points in the generated sentence structure. For this to happen, a number of evaluation mechanisms are needed, defining criteria to assess each sentence on its relevance (will this be interesting to a human user?), clarity (will this be understandable by a human user?), or creativity (will this step be a creative milestone [16] where the design shifts?).
- *non-sequentially*, summarizing the explanation starting from the most important points regardless of when they were performed in the generative process. Indeed, it is possible to start by presenting a description (visual, textual, or otherwise) of the final artifact and backtracking some of its most interesting elements on points in the generative process where those happened. Moreover, tropes such as sports game summaries can be used as inspiration, presenting the main outcomes of the generative process first (as non-sequential highlights) followed by a longer form of the sequential narrative regarding how generation progressed from unformed to fully formed content.

B. Use Case 2: Black-box PCG System

In a black-box PCG system such as a level generator or world builder, the AI assistant needs to (a) share a common language of design with the human designer, (b) communicate its current understanding of this language, and (c) update this understanding in response to designer feedback. Essentially, this is the notion of establishing 'common ground' as forwarded and explored by Herb Clark [31]. The designer will provide input into the black-box AI assistant with the goal of receiving a full or partial design from the system. For example, assume that the AI assistant aims to generate the 3D geometry for a city similar to CityEngine [32]. The input will be a size of the land with topology on which to generate the city and a set of parameters that are used in the construction of the road and transit structures, building designs, and placement of buildings in blocks along streets (as this is how humans build cities). There is a lot of information to convey and a nearly infinite way to construct cities; however, cities are created all the time—even virtual ones. The black-box PCG system will need to be clear about how it takes that input and ultimately how that connects to and affects the output. It is the transformation by the AI from input to output that needs to be explained and this is where common ground is leveraged.

A designer first working with an AI assistant could spend a great amount of time probing the system with variations and developing a mapping (or model) of how changes in input impact output (learning by observation). However, that is quite tedious and a more abstract, explained transformation process would be faster to comprehend and work with for a designer. A human may direct an AI assistant to build a city in the 'American style'; knowing that this means a city laid out in generously-sized square blocks with most streets having simple intersections is an easy and powerful way to produce a desired design. Ultimately, this is a direction for an agreed concept in the transformation process of the technique. There are a lot of details needed to produce that design, which are encapsulated in a specific design concept. Mechanisms that build and update that common ground language and mapped meaning need to be added to the techniques inside the black-box. For reactive techniques, the artifacts of training data and the process of machine learning may need to be included in some form to facilitate explanation of the internal mechanisms.

Imagine a black-box AI assistant using a DNN to recognize ideal topology for road placement, which is then placed by a set of construction rules biased by a provided set of city road layout examples filtered by design language labels. In order to build *common ground*, the AI assistant will need to interactively show the designer how the provided land and topology are perceived and how its prior examples are used to generate roads based on the language provided, as well as how these concepts may roll-up hierarchically in the system. This may involve keeping connections to the training data used in the deliberative creation process of reactive techniques in the system. Thus, the system reveals enough information to allow the induction of a model of the AI in the designer. When a mapping of the designer's internal model is connected to a correct induction of the AI assistant's internal model, common ground is established by sharing a language, an understanding, and the ability to update both sides easily. The key challenge is how and what to share to build that model in the designer's mind without exposing them to the potentially massive amounts of data used to train the network and used by the system for making decisions. Induced models in humans can be tested by predictive capability and accuracy.

On the XAID spectra, black-box PCG AI assistants for designers require the most explainability as they involve learning, recognizing, and extending patterns to create content the subtleties of which a designer will want to understand and work on together with the AI. On initiative, these systems are likely to be *on-task* colleagues or have high-functioning autonomy. On domain overlap, as the example given in Section V-C, creating content with an AI will have high overlap, but *on-demand* and often *turn-based*. The PCG AI assistant and

the designer work closely together, refining until the desired content is produced. The designer provides the vision, the AI provides capabilities, and they merge that into the creation.

C. Use Case 3: Black-box NPC Behavior System

In a third use case, imagine an enemy NPC behavior controlled by a trained DNN. The goal of the game designer is to see whether the NPC behaves as intended in a new game level (in our case, an infirmary). Since the network encodes complex NPC behaviors in an opaque way, an XAID system should be able to help the designer to better understand the NPC AI. We design this XAID task to be *observable* and *passively* awaiting the designer's request to provide insights on how the NPC will behave.

Two types of information are of particular importance to the designer. First, given the layout of the level, what is the likely distribution of actions the NPC will take? For example, at the entrance of the infirmary, how often will the NPC walk straight inside, turn left to interact with another NPC, or turn right and avoid the level altogether? This distribution can be approximated through sampling, i.e. letting an NPC play a certain level multiple times with slight variations in starting position, etc.. If it is crucial for the player to encounter this NPC and the latter has a high chance of leaving, the designer needs to be informed and be provided with a reason why this happens and how to correct it. Through a mixed-initiative approach, the system could also suggest changes to the environment that would make the desired behavioral outcome more likely.

Second, given a particular NPC action, what are all the possible situations that can lead to this action? If the NPC sometimes has the unexpected behavior of shooting at a window, it would be helpful for the XAID to show all the situations where this will happen. By providing a full list of scenarios that will lead to a particular action, this feature will make the NPC behavior system more predictable and thus may increase designer's trust in the behavior system. Methods such as feature visualization and attribution (Section II-A) can give insights to what stimulus the network will react to. To the best of our knowledge, however, there is currently no approach that can do all of this in an automated way.

Given the large number of possible actions and/or situations, similar to highlights in white-box PCG systems, a good design guideline for XAID is to highlight the unexpected and reduce the visibility of the common ones. A key open challenge to providing both types of information to a human designer is how to design the reward function for the NPC.

VII. Open Challenges

In this section we point out some of the open challenges in providing useful XAID in relation to both white-box and black-box systems, as well as their combination.

A. White-Box Systems

An open challenge in providing useful XAID is how to fit the entire process of the white-box system into something that is compact and yet sufficient for designers. Similar to how a black-box ML model can show all of the training data (Section VII-B), a white-box model can explain (i.e. narrate) the sequence of all actions that it takes (including iterations within loops). The challenge is how to cluster or omit activity reports that are less relevant for the designer to know. Some of the actions reported may be too 'esoteric' (i.e. tied to the system's internal method of understanding the world or producing new artifacts) for the user to understand. In order to create 'highlights' as noted in Section VI-A, the challenge of evaluating subjective notions such as *interestingness* or *relevance* may require a computational model of the individual designer [33]. Moreover, such criteria might need to operate beyond the horizon of a single sentence; the whole narrative (sequence of sentences) must be produced before the most interesting points within it are chosen in a post-processing step. In addition, the concrete features of the final artifact (be it game content, NPC behavior, etc.) may also be relevant for this post-processing.

B. Black-Box Systems

While different techniques are now emerging that can give some insights into the working of black-box systems such as neural networks (surveyed in Section II-A), they can currently only help to interpret a model but lack full explainability. The more complex these models become, one can wonder if it will ever be possible to fully explain their inner workings.

Meaningful abstraction from base provenance can be difficult. While it is possible to show all the training data or even clusters of training data for explanation, this process may overwhelm a user and fail to induce a model of understanding. Providing proper, meaningful and likely hierarchical abstractions of training data and the transformation into learned models is an open challenge. While the core question that leads to understandable AI is 'why?', the answer should come from introspection, which is an open challenge for many techniques, especially black-box methods.

C. Combined approaches

While explaining white-box and black-box systems is difficult in the context of XAID, understanding complex AI systems with many parts and multiple techniques becomes an even more challenging problem; understanding the sum is not the same as understanding the parts. An important future direction will be the development of dialog and concept grounding, building common ground between AI and designers.

VIII. Conclusions

In conclusion, we proposed the new research area of eXplainable AI for designers (XAID), to help game designers better utilize AI and ML in their design tasks through co-creation. Our position is that, in order to make usable and efficient XAID systems, we need to build on understandings of both algorithmic properties of the underlying AI techniques and the needs of human designers. We mapped the space of XAID with three axes—the spectra of explainability, initiative,

and domain overlap—and illustrated our approach through three specific use cases. Based on a deeper analysis into use cases, we identified key open challenges.

ACKNOWLEDGEMENTS

This article extends the work performed during the 2017 Dagstuhl seminar 17471 on "Artificial and Computational Intelligence in Games: AI-Driven Game Design". An initial report on the topic, titled "Explainable AI for designers", can be found in [34]. We would like to thank all of the researchers who contributed their ideas in writing the Dagstuhl report; their insight has led to this publication.

REFERENCES

[1] A. Nguyen, J. Yosinski, and J. Clune, "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 427–436.

[2] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep inside convolutional networks: Visualising image classification models and saliency maps," *arXiv preprint arXiv:1312.6034*, 2013.

[3] A. Nguyen, J. Clune, Y. Bengio, A. Dosovitskiy, and J. Yosinski, "Plug & play generative networks: Conditional iterative generation of images in latent space," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 3510–3520.

[4] D. Erhan, Y. Bengio, A. Courville, and P. Vincent, "Visualizing higher-layer features of a deep network," University of Montreal, Tech. Rep. 1341, Jun. 2009.

[5] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *Proceedings of the European conference on computer vision*. Springer, 2014, pp. 818–833.

[6] R. C. Fong and A. Vedaldi, "Interpretable explanations of black boxes by meaningful perturbation," *arXiv preprint arXiv:1704.03296*, 2017.

[7] O. Boz, "Converting a trained neural network to a decision tree dectext - decision tree extractor," Ph.D. dissertation, Lehigh University, 2000.

[8] C. Olah, A. Satyanarayan, I. Johnson, S. Carter, L. Schubert, K. Ye, and A. Mordvintsev, "The building blocks of interpretability," 2018, [Online; accessed 13 May 2018]. [Online]. Available: https://distill.pub/2018/building-blocks/

[9] O. Biran and C. Cotton, "Explanation and justification in machine learning: A survey," in *Proceedings of the IJCAI Workshop on Explainable Artificial Intelligence*, 2017.

[10] C. Lacave and F. J. Díez, "A review of explanation methods for bayesian networks," *The Knowledge Engineering Review*, vol. 17, no. 2, pp. 107–127, 2002.

[11] B. Seegebarth, F. Müller, B. Schattenberg, and S. Biundo, "Making hybrid plans more clear to human users-a formal approach for generating sound explanations," in *Proceedings of the International Conference on Automated Planning and Scheduling*, 2012.

[12] J. Bidot, S. Biundo, T. Heinroth, W. Minker, F. Nothdurft, and B. Schattenberg, "Verbal plan explanations for hybrid planning," in *Proceedings of MKWI workshop on Planung/Scheduling und Konfigurieren/Entwerfen*, 2010.

[13] M. Fox, D. Long, and D. Magazzeni, "Explainable planning," in *Proceedings of the IJCAI Workshop on Explainable Artificial Intelligence*, 2017.

[14] S. Rosenthal, S. P. Selvaraj, and M. M. Veloso, "Verbalization: Narration of autonomous robot experience," in *Proceedings of the IJCAI conference*, 2016, pp. 862–868.

[15] R. Straatman and A. Beij, "Killzones ai: dynamic procedural combat tactics," in *Game Developers Conference*, 2005.

[16] G. N. Yannakakis, A. Liapis, and C. Alexopoulos, "Mixed-initiative co-creativity," in *Proceedings of the Foundations of Digital Games Conference*, 2014.

[17] D. Novick and S. Sutton, "What is mixed-initiative interaction?" in *Proceedings of the AAAI Spring Symposium on Computational Models for Mixed Initiative Interaction*, 1997.

[18] A. Liapis, G. N. Yannakakis, and J. Togelius, "Sentient sketchbook: Computer-aided game level authoring," in *Proceedings of the Foundations of Digital Games Conference*, 2013.

[19] G. Smith, J. Whitehead, and M. Mateas, "Tanagra: Reactive planning and constraint solving for mixed-initiative level design," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 3, 2011.

[20] M. Cook, J. Gow, and S. Colton, "Towards the automatic optimisation of procedural content generators," in *Proceedings of the IEEE conference on Computational Intelligence and Games*, 2016.

[21] M. Cook and S. Colton, "Ludus ex machina: Building a 3d game designer that competes alongside humans," in *Proceedings of the International Conference on Computational Creativity*, 2014.

[22] S. Colton, J. Halskov, D. Ventura, I. Gouldstone, M. Cook, and B. Perez-Ferrer, "The Painting Fool sees! New projects with the automated painter," in *Proceedings of the International Conference on Computational Creativity*, 2015.

[23] B. F. Malle, *How the mind explains behavior: Folk explanations, meaning, and social interaction*. Cambridge, MA: MIT Press, 2004.

[24] P. Pu and L. Chen, "Trust building with explanation interfaces," in *Proceedings of the 11th international conference on Intelligent user interfaces*. ACM, 2006, pp. 93–100.

[25] D. Kahneman, *Thinking, fast and slow*. Farrar, Straus and Giroux, 2011.

[26] A. Abdul, J. Vermeulen, D. Wang, B. Y. Lim, and M. Kankanhalli, "Trends and trajectories for explainable, accountable and intelligible systems: An hci research agenda," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2018.

[27] F. Doshi-Velez and B. Kim, "A roadmap for a rigorous science of interpretability," *arXiv preprint arXiv:1702.08608*, 2017.

[28] T. Miller, P. Howe, and L. Sonenberg, "Explainable AI: Beware of inmates running the asylum," in *Proceedings of the IJCAI Workshop on Workshop on Explainable Artificial Intelligence*, 2017.

[29] M. Shaker, N. Shaker, and J. Togelius, "Ropossum: An authoring tool for designing, optimizing and solving cut the rope levels," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2013.

[30] J. Charnley, S. Colton, M. T. Llano, Rodriguez, and J. Corneli, "The FloWr online platform automated programming and computational creativity as a service," in *Proceedings of the International Conference on Computational Creativity*, 2015.

[31] H. H. Clark, R. Schreuder, and S. Buttrick, "Common ground at the understanding of demonstrative reference," *Journal of Verbal Learning and Verbal Behavior*, vol. 22, no. 2, pp. 245 – 258, 1983.

[32] ESRI, "City engine," 2018. [Online]. Available: http://www.esri.com/software/cityengine

[33] A. Liapis, G. N. Yannakakis, and J. Togelius, "Designer modeling for personalized game content creation tools," in *Proceedings of the AIIDE Workshop on Artificial Intelligence & Game Aesthetics*, 2013.

[34] P. Spronck, E. André, M. Cook, and M. Preuß, "Artificial and Computational Intelligence in Games: AI-Driven Game Design (Dagstuhl Seminar 17471)," *Dagstuhl Reports*, vol. 7, no. 11, pp. 86–129, 2018.

Using a Team of General AI Algorithms to Assist Game Design and Testing

Cristina Guerrero-Romero, Simon M. Lucas and Diego Perez-Liebana
School of Electronic Engineering and Computer Science
Queen Mary University of London
London, UK
{c.guerreroromero, simon.lucas, diego.perez}@qmul.ac.uk

***Abstract*—General Video Game Playing (GVGP) has become a popular line of research in the past years, leading to the existence of a wide range of general algorithms created to tackle this challenge. This paper proposes taking advantage of this research to help in game design and testing processes. It introduces a methodology consisting of using a *team* of Artificial General Intelligence agents with differentiated goals (winning, exploring, collecting items, killing NPCs, etc.) and skill levels. Using several agents with distinct behaviours that play the same game simultaneously can provide substantial information to influence design and bug fixing. Two methods are proposed to aid game design: 1) the evaluation of a game based on the expected performance in the behaviour of each of the agents, and 2) the provision of visual information to analyse how the experience of the agents evolves during the play-through. Having this methodology available to designers can help them decide if the game or level under analysis fits the initial expectations. Including a *Logging System* can also be used to detect anomalies while the development is still at an early stage. We believe this approach allows the flexibility and portability to be easily applied to games with different characteristics.**

***Index Terms*—methodology, General Artificial Intelligence, automatic testing, game design, team of agents**

I. Introduction

Games evolve during their development process, both in terms of implementation and design. New ideas are put into practice during the production phase and need to be tested quickly and efficiently. While using human play-testing is a broad practice, there is no denying that this impacts the company in terms of resources (human and technological), time and money. Agent-based testing is a suitable alternative for automatic game testing, and there is a prolific body of work (as described later in this paper) that uses game specific agents to evaluate content and agent behaviour. Although this can provide some advantages for fast and reliable testing, the design and implementation of these specific agents may not be adaptable enough to the changes designers and developers are regularly introducing. The use of general algorithms, on the other hand, provides a level of generalisation, portability and flexibility that cannot be matched by game-specific ones.

This paper proposes a methodology consisting of a *team* of Artificial General Intelligence (AGI) agents with differentiated goals to aid the design and testing processes during the development of a game. Each of the agents forming the team has its own objective (winning, exploring, collecting items, killing NPCs, etc.) and skill level. This set up provides a flexibility that would not be possible using just one. The designer can choose the agents to run and set their expected targets of performance. Each of the specialists selected plays the game under evaluation focusing on its own goal and, as a result, a Logging System and two types of reports are generated. The first one gives information about how accurate the estimated performance for each of the behaviours is, compared to the actual results. The second one shows a graph that provides visual feedback of how certain information of the game is retrieved by the agents and evolves during the play-through.

This team of general agents is meant to respond to changes and updates across multiple dimensions of game design:

*1) **Rules**:* The base of every game. Making any change to the rules can trigger unexpected outcomes and affect other rules in a way the designer did not plan to. General agents are independent of the rules so they do not need to be adjusted when they change to be able to check that everything is working as it should be. It grants the possibility of carrying out immediate testing to detect anomalies as soon as they appear. It provides flexibility to the methodology, and it is one of the core ideas of the approach proposed in this paper.

*2) **Levels**:* Where the action takes place. They shape how the game is presented to the player. It includes increasing the level of difficulty, reachable areas and distribution of the elements of the game: the proportion of enemies by stages, collectable items dispersed uniformly, etc. Reports from each one of the general agents after they have played a certain level can provide the information needed to check that these points are covered as expected. An example would be analysing the evolution of the number of Non-Player Characters (NPCs) eliminated by the *Killer* (III-B8). The designer should be able to notice peaks and an abrupt increase in the numbers in those stages where they expect a big confrontation. If the play-through graph does not present those peaks, the level should be reviewed and fixed to work as desired.

*3) **Non-Player Characters (NPCs)**:* NPCs' performance and interactions with the player have a big impact on the experience while playing the game. Any update on their implementation should be tested and their impact on player experience analysed and measured. Analysing the general agents' reports and behaviour could provide an insight of this. For example, checking the number of deaths vs. kills of the

978-1-5386-4359-4/18/$31.00 ©2018 IEEE

Killer after a change is done to the NPCs, comparing the difference on percentage of life lost between two *Killers* with known disparate level of mastery, or tracking the whereabouts of the NPCs (logging similar information to the one measured for the *team*) for behavioural checking.

*4) **Game Parameters**:* Even small updates in the parameters can have a big impact in the game. An example is updating the height of the jump of the avatar: if it is set to a very low value, they might not be able to reach some areas of the game, affecting the exploration. Analysing the information provided by the agents can give a clue to know if the parameters are set properly. In this example, the percentage of the exploration reached by the *Map explorer* (III-B2) when the height of the jump is modified could increase or decrease abruptly.

The proposed methodology works within the game. If using algorithms specific to the game, every time any of its elements is changed the algorithms would need to be updated as well. Having general algorithms, with general goals independent from the rules, implies that they do not have to be modified every time a change is done. This is considered to be one of the biggest strengths of the method proposed in this paper. Another discussed benefit is being able to use the same algorithms, without modifications, in different levels of the same game as they are being created. Because of the general goals, the heuristics would not need to be updated to fit the specifications of a new level. It would allow checking if it fulfils the expectations almost immediately after being included in the game. Finally, as Section II-A states, there are many types of GVGP algorithms, which provide a wide range of options depending on the technology, characteristics and implementation of the game considered. An example would be the availability of a forward model or not.

The use of General AI does not mean that some game-specific tweaks should not be added to improve the heuristics performance, as long as the main general goals are not changed. The strength of the proposed approach is based on the generality, flexibility and robustness concepts of general AI, which can adapt to significant changes in the game design.

This paper provides an overview of General AI frameworks and automatic testing approaches in Section II, followed by the description of the proposed methodology in Section III. Section IV describes the limitations of this approach, and conclusions and possible extensions are detailed in Section V.

II. Background

A. General Video Game Playing

General Video Game Playing (GVGP) aims to develop algorithms capable of playing video games without having prior knowledge about them, with mere access to the state of the game and the available actions [1]. The interest in the research in this area has grown in recent years.

The most common techniques to tackle the problem, with different implementations, are the use of *Reinforcement Learning* (RL), *Tree Search* and *Evolutionary Algorithms*. In order to provide the resources to be able to develop and study these different approaches, a series of frameworks have been created. The main open-source frameworks available to the research community are the Arcade Learning Environment (ALE) [2], the General Video Game AI Framework (GVGAI) [3], OpenAI Gym [4] and Project Malmo [5], among others. These frameworks share the desire of encouraging the study of the Artificial General Intelligence (AGI) but have different characteristics, leading to the creation of numerous types of algorithms, which keep growing and being improved.

One of the first general frameworks is **ALE**, a testbed for comparing and evaluating planning and learning algorithms, providing an interface to Atari 2600 games, like *Space Invaders* and *Ms Pac-Man* [2]. Most of the research carried out using this framework has focused on *Reinforcement Learning*, as Mnih *et al.* work [6]. They showed how using Deep Q-Networks (DQN) receiving only a screenshot and the game score as inputs through a set of 49 games it was possible to achieve a level of performance comparable to a professional human player in many of the games tested. The environment also allows the use of planning algorithms, but the research in this area using this framework is very rare. A reason for this could be the complexity in finding heuristics general enough to have good performance over all the games [7].

The **GVGAI Framework** has been used for the ongoing GVGAI Competition since it was run in 2014 [3]. The games used to benchmark the algorithms are described in the Video Game Description Language (VGDL) [8], originally implemented in *Python* by Tom Schaul [9]. It allows the implementation of single and two-player 2D Arcade games.

Providing a forward model that allows agents to foresee the possible states originated by taking any of the available actions, the first competition encouraged the submission of single player planning algorithms. Over the years, the competition has been expanded to cover other novel areas of general AI, releasing two-player [10], level [11] and rule generation [12], and a learning track [13], which removes the availability of the forward model and provides a screen capture to promote research on other learning algorithms.

The algorithms developed to work in this framework and submitted to the competition are very assorted: several variations of Monte Carlo Tree Search (MCTS) [14], including its Open-Loop variations (OLMCTS) that works better in stochastic environments, and evolutionary algorithms, like Rolling Horizon Evolutionary Algorithm (RHEA) [15] and Random Search (RS). Two algorithms that, so far, have shown best overall performance, and therefore, have been claimed winners of some of the competitions, are Adrien Couëtoux's Open Loop Expectimax Tree Search (OLETS) [3] and Joppen *et al.*'s YOLOBOT [16]. An insight of the framework, its wide use, algorithms implemented and a complete list of the winner algorithms per year can be found in a recent survey [17].

Another popular toolkit is **OpenAI Gym**, oriented to test *Learning* approaches providing a common interface for a collection of environments based on pre-existent RL benchmarks [4]. It includes, among others, *Atari*, which uses ALE. This collection grows over time.

In contrast with other frameworks, it provides an abstrac-

tion for the environment instead of the agent and does not provide a hidden test set. *OpenAI* encourages peer review and collaboration by sharing the code and a description of the approach followed, instead of arranging a competition between the algorithms. The framework focuses on both the performance of an algorithm and the amount of time it takes to learn. It keeps a strict version number scheme every time a change is made in an environment.

Finally, **Project Malmo** is a platform built on top of *Minecraft* designed to support AGI research in reinforcement learning, planning, multi-agent systems, robotics and computer vision [5]. In this framework, agents are exposed to a 3D environment with complex dynamics that provides the experimenters with the tool to set complicated tasks. In 2017, the *Malmo Collaborative AI Challenge*[1] was run using this environment to encourage the research in collaborative Artificial Intelligence. The goal of the competition was to create agents capable of learning to achieve high scores when working with a range of both artificial and human partners.

The existence of these (and other) frameworks implies that there is a huge variety of algorithms with different characteristics, weaknesses and strengths, at everyone's disposal. There is a large and active community of researchers working on improving those general algorithms and creating new ones.

B. Automatic Testing and AI Assisted Game Design

When creating a game or adding new levels to it, they should go through a testing process to make sure their characteristics are aligned with the expectations and that no bugs are affecting the gameplay. The Quality Assurance (QA) of the games is usually carried out manually by members of the development team or game testers. Automated testing aims to facilitate the QA by using automatic processes. They are generally game-dependent, which is a big limitation as they should be implemented specifically for the game under development. This paper proposes a methodology general enough to be easily adaptable to any game, without having to invest much time in game-specific setting ups.

Intrinsic motivation refers to a series of physical needs that motivates a certain behaviour without the direct existence of an external reward like the score. S. Roohi *et al.* state how the emerging field of simulated-based game testing looks promising [18]. Being able to use simulated agents instead of human players to provide feedback during the game design process can increase the speed and reduce the costs. These authors review the existing literature on intrinsic motivation in player modelling, focusing on simulation-based game testing. They come to the conclusion that its application to automatic testing is sparse and hope that their work would provide new ideas to the research community. This paper takes this inspiration and suggests using a series of agents (not necessarily just intrinsically motivated) for simulation-based game testing.

In [19] Holmgård *et al.* present an approach for automated playtesting using *archetypal generative player models* called *Procedural Personas*. In this work, they use a variation of Monte Carlo Tree Search (MCTS) where its Upper Confidence Bound (UCB) equation is adapted by evolution to be able to create players with differentiated goals and behaviours. They create four *Procedural Personas* (*Runner*, *Monster Killer*, *Treasure Collector* and *Completionist*) to play their test game *Minidungeons 2* focused on four different primary objectives. These are, in order, reaching the exit, killing enemies, collecting items and consuming any game object that is possible to be collected or killed. All of them were also given a secondary goal: reaching the exit as quickly as possible for the *Runner* or just being able to reach it, for the rest of them.

The authors ran a series of experiments to compare the performance between the evolved personas and the baseline algorithms and to test how different they interact with the environment. The results show how all these evolved personas perform better than baseline UCB1 ones regarding the computational time required to reach the exit and, therefore, finishing the levels. They notice how these evolved personas, even when all managed to reach the end of the level, had differentiated play-styles depending on their primary goal and were affected by the patterns of the level played. They discuss how these personas with differentiated behaviours can be used for level evaluation, either providing feedback to a human game designer or assisting the improvement of automatically generated levels, driven by their distinct play-traces. Because they are oriented to provide useful feedback to the game or level designer, they argue how they should define the utility functions to fulfil the priorities of the design.

The methodology presented in this paper is inspired by this work but aims to have a more general and portable approach, capable of being applied to several different games without having to design specific types, or utility functions to fit the game under consideration. We believe that extending the idea to use general agents, developed with general goals that can be applied to several different games, can provide significant advantages. A *team* of pre-defined types with general goals and approaches gives the designer the chance to choose which agents fit the characteristics of the game.

S. Nielsen *et al.* used the *Relative Algorithm Performance Profile* (RAPP) approach to estimate the quality of a certain game based on the performance of general agents [20]. They compared the performance between known algorithms in a range of hand-designed, mutated and random generated VGDL games. Their premise argued that a game that has a high skill differentiation is likely to be a good one. Despite the results backing their hypothesis, complexity does not necessarily infer quality and, by its own, this approach is not able to provide further information about the game under evaluation.

The evaluation that this paper proposes is based on the performance of the general agents, but using a different approach. Although RAPP can be used to make the methodology stronger, there are some important differences to highlight. Firstly, in [20] they used seven different general algorithms, including an *explorer*, but they only based their performance on the winning rate and difference of score. In our case, the

[1] https://www.microsoft.com/en-us/research/academic-program/collaborative-ai-challenge/

agents used, if with distinct goals, are not compared between them. However, they provide particular information about their own play-through and performance based on each of their objectives. Also, our methodology is expected to provide deeper feedback and richer information than a mere state of the good/bad quality of the game.

T. Machado *et al.* built the Computationally Intelligent Collaborative EnviROnment (*Cicero*) [21], which is a general-purpose AI-assisted tool for 2D tile-based game design. It was built on top of the GVGAI framework, assisting in the creation and development of VGDL games. It provides a game editing mechanism to add the sprites and rules that form the game and includes a *mechanics recommender*. It suggests certain sprites and rules based on the ones added. It also grants an automatic testing feature that shows game rule statistics in real time and a level visualisation. To test the game, it is possible to either play it manually or select one of the general agents available in the framework. Running the game with a general agent provides heat-maps of the player and the NPCs. Also, during the automatic gameplay, a list with the different rules and the stats for each of the interactions of the game are shown. Cicero was expanded to include *SheekWhence*, a *retrospective analysis tool for gameplay session* [22]. This extension includes a recording of the gameplay to analyse the sequence of events, being able to go forward and backwards in the session. Its limitation is that it is very oriented to VGDL and the GVGAI framework, impeding its application to a wider range of games. Also, although general algorithms are used for evaluation, it is not taking advantage of most of the information that could be extracted from their play-through to provide richer information to the designers. Moreover, these agents' ultimate goal is winning, so their behaviour is not as assorted as including a *team* with different objectives.

III. General AI Team to Assist Game Design

A. Overview

This paper proposes a methodology capable of assisting the design and testing process during the development of a game. The evaluation uses a *team*; a series of General AI algorithms with differentiated goals (Section III-B). Each one of the agents plays and behaves differently within the same game. Extracting certain information from their play-through, and having the right tools to interpret it can help the designer. They can check if they are on the right path, or if a change needs to be carried out to get aligned with the expected outcome. In contrast with meta-heuristic approaches, where a heuristic is involved in deciding which type of agent is run depending on the state of the game, in our case, all the agents play the game simultaneous and independently.

The methodology needs a series of entities to work. The **Designer** is the final user and responsible for the game. They want to make sure that the content (game or level) under development fits the expectations of the design without errors. They would provide the part of the **Game** and set up the processes required for its evaluation. Three types of outputs are generated after the methodology is applied to evaluate the game, two of which are reports. Firstly, the **Target Reports** provide the results of evaluating the game based on each of the agents' behaviours, compared to expected targets. These targets are set before the tests are run. Next, the **Visual Reports** provide visual information about the evolution of the information retrieved by each of the agents during their play-through. This information is presented in a series of graphs. Lastly, the **Logging System** records the logs resulting from the algorithms' play-through to provide support for testing and debugging (Section III-D).

The main steps of this methodology are as follows:

1) ***Setting up the team****:* There is a range of general agents of different types and with a range of skills. The designer can choose, and optimise, the ones they believe fit the characteristics of the game and design expectations. They can also set an expected performance for each of the agents.

2) ***Integrating the game****:* The methodology focuses on being portable and flexible enough to be used with different games. However, it is needed to set it up to be able to run the algorithms, extract information from their play-through and record the metrics in the Logging System.

3) ***Evaluation process****:* The types of agents and skills picked by the designer are run a certain number of times in the game provided. Each of the agents' gameplay logs a series of metrics and errors triggered to be able to have detailed information about what happened.

4) ***Generating reports****:* The information provided by each of the agents (Section III-B) is processed to generate the two different type of reports presented in Section III-C.

B. The Team

This paper proposes using a series of general algorithms with differentiated goals, capable of playing a game focusing on their specific objectives. Differentiating the heuristics in General Video Game Playing was introduced by C. Guerrero-Romero *et al.* in [23], and some of the members of the suggested *team* have been inspired by their work.

The inspiration also comes from R. Bartle's *player types* [24]. This work presented four approaches to play MUD games, showing how the same game can be played in various ways based on the motivation of the players, leading to distinct behaviours. Even when this work is specific for MUDs, it has been a reference to find types applicable to different games. Recently, N. Yee has developed a player motivation profile based on data from more than 250.000 players, coming up with 6 main differentiated clusters of gaming motivations [25].

A *team* of agents focused on different tasks provides a flexibility that would not be possible using just a specific one. The general objectives presented in this section cover different aspects, which could be present, or not, in a game. The designer can accommodate the methodology to adapt their intentions and needs by including the agents to fit its characteristics. The following is a non-exclusive list of agents proposed to form the team, their targets and the information to provide:

1) ***Winner****:* Focused on winning the game; maximising the score when a winning state is not immediately reachable. The information provided by an agent of this type can be the number of wins, game ticks to victory, or strategy followed when there is more than one option available.

2) ***Map explorer****:* Focused on covering the reachable areas as much as possible. The information provided by an agent of this type can be the number of different positions of the map visited, the total percentage of the map explored, or game ticks required to finish the exploration.

3) ***Novelty explorer****:* An alternative for an exploratory agent is considering states instead of positions; going through as many different game states as possible and providing this number as a result. It is related to the *Novelty* appraisal common in intrinsically motivated agents in AI [18]. Also, the inspiration for this kind of agent comes from the work done by M. Bellemare *et al.* in [26]. The authors proposed connecting the information gained through the learning process and count-based exploration, which guides agents' behaviour to reduce uncertainty. This approach is designed to explore the environments more practically and efficiently.

4) ***Curious****:* Focused on interacting as much as possible with the elements of the game, always prioritising those that have not been interacted with before. The information provided can be the number of elements interacted with, actions triggered when they happened, or game ticks required to interact with the different elements of the game.

5) ***Competence seeker****:* Based on the model of *empowerment* of intrinsic agents, which denotes the degree of control the agent feels having over the environment [18]. It is related to the amount of information the agent is capable of collecting when a series of actions are performed. It can provide information about the level of expertise gained during its play-through.

6) ***Record breaker****:* Focused on maximising the score and solving puzzles, without paying attention to the chances of winning the game. The information provided can be the number of points obtained, puzzles solved or game ticks required for these.

7) ***Collector****:* Focused on collecting the items available in the game. The information provided can be the number of items collected, counts per type of item, or game ticks required to collect the different items present in the game.

8) ***Killer****:* Focused on removing from the game as much Non-Player Characters (NPCs) as possible. The information provided by an agent of this type can be the number of NPCs killed, the number of times killed by an NPC, counts per type of NPC encountered, or game ticks required to kill all the enemies present in the game.

9) ***Risk analyst****:* Focused on analysing the level of risk during the play-through and taking actions to maintain it at a certain level chosen by the game designer. A low-risk agent would tend to avoid situations where the chances of losing the game are high, like bumping into a hoard of enemies or complex areas. A high-risk agent would tend to do the opposite and jump into dangerous situations. The information provided by an agent of this type can be the risk percentage predicted at every moment, the number of deaths, NPCs killed, obstacles overcame or game ticks until losing the game.

10) ***Semantic****:* Focused on tasks related to linguistics, as coaching the dialogue of the game or making sure the narration flows and is consistent. The information provided by an agent of this type can be the estimated quality of the dialogues, the number of possible outcomes depending on the choices and the level of consistency of the narrative.

11) ***Scholar****:* Focused on learning the outcome of the actions available, taking as much knowledge about the game as possible. The information provided by an agent of this type is the percentage of accuracy of the knowledge gained during the duration of the gameplay. As it is needed to have concrete information about the rules and outcomes of the interactions with the game to be able to check the quality of the predictions, the generality of this type of agent is improbable. However, an agent with this kind of objective is an interesting addition to the team as it can be used to detect anomalies during gameplay. There is a high chance that an agent focused on this kind of task finds unexpected rules or bugs on the existent ones that should be fixed.

C. Assisting Game Design

Two different types of reports are provided in order to check the validity of the design of the game.

The first kind is **Performance-target based reports**, thought to evaluate the game based on the expected performances in the behaviour of each of the agents. In the experiments carried out in the GVGAI framework for the work presented in [23], results for same heuristics algorithms showed a clear distinction depending on the type of game. A clear example is the results obtained using the Exploration Maximization Heuristic (EMH). Algorithms using this heuristic focused on maximising the exploration of the level. Their performance was calculated by obtaining the percentage of the level explored dividing the number of different positions visited, by the total. In completely accessible maps in games like *Butterflies*, the agents using the EMH ended up with an average percentage of performance higher than 80% in most of the cases. Whereas, in games with large maps, or where a series of steps were needed to unlock the access to the different areas, like *Roguelike*, any of those agents got an average higher than 45%[2]. The presence of these differences on performance can be used in designer's benefit, providing an estimation of performance that agents should achieve depending on the type of game designed.

Before running the *team*, the designer would be able to choose the agents considered appropriate for the game under evaluation and to set an estimated desired percentage of performance for each of them. After a series of runs carried out by each of the agents, an error for the expected values would be obtained and returned, to inform if there is an agreement between the ideal values and the reported ones. For example, if

[2]This percentage was not explicitly mentioned in the paper, but it has been taken from the same results obtained in those experiments

a designer plans a game to be easily accessible but challenging to win, they would assign a high desired value to the *Map explorer* and a low value to the *Winner*. After several runs of the agents, the errors would be reported by calculating the difference between the targets and the real values. With this information, they would be able to check if the design matches the expectations, or how distant the values are.

The second type of information retrieved can be easily interpreted by the designer in the form of **Visual reports**. These are meant to provide graphs that analyse how the information retrieved by the agents (number of different positions or states visited, number of elements interacted with, etc.) evolves during the play-through. The designer should be able to extract and conclude interesting information about their game by analysing the shape and evolution of the plotted values. A continuous trend means that the agent is capable of getting information without many impediments, improving uniformly. On the contrary, if the growth is stuck for a period of time, it either means that there is nothing more to be discovered, all targets of the agent have been reached, or that there is an obstacle (or a series of obstacles) blocking the agent to achieve its goals. Let's take a possible play-through graph obtained for the *Map explorer* as an example. It could show a uniform growth to a certain point, keep still for a while to end up increasing uniformly again. This shape could be interpreted as follows: the map of the game is divided into two areas and an action from the player is required to progress in the game.

This method could also be used to analyse the distribution of different elements of the game. In the example of the *Collector*, the growth of the graph would show peaks in those areas where there are several items to collect.

D. Logging System

The *Logging System* keeps track of the information resulting from running each of the agents: position by time, actions, elements interacted with, responses triggered, etc. These logs can help to detect anomalies and broken states of the game.

M. Nelson proposed seven strategies to extract information from the game [27]. In [28], V. Volz *et al.* gather a list of measures envisioned to be included in the GVGAI framework to extract information from the gameplay. They differentiate between agent-based, interpreted and direct and indirect loggable measures. Because of the generality of the framework this list was collected for, it could be taken as a reference to use in this methodology. Having a *team* of agents, instead of a unique one, can cover more game states, allowing to trigger errors that would be difficult to catch otherwise.

E. Variations

Same algorithms with different parameters have different strengths. The *team* can include several versions of the algorithms with same objectives, but different levels of mastery, based on those parameters. There are several existing methods to be able to arrange a series of algorithms by measuring their performance, used for several competitions and online rankings. The most distinguished ones are the *Bayes Elo system* [29], *Glicko* [30] and *TrueSkill* [31], the skill rating system used in Xbox Live and recently extended. The designer can be given the opportunity to choose between differentiated skilled agents and even perform Relative Algorithm Performance Profiles checks (II-B). This enlargement allows an even bigger range of choices and richer information available.

Another possible extension can be to take into consideration the information retrieved by all agents as a whole and study the correlations between them. The designer can choose which agents' information combine to obtain greater levels of granularity.

IV. Limitations

The approach proposed in this paper has a clear strength, but there are a series of limitations, presented in this section.

The time needed to perform the evaluations should be taken into consideration to arrange enough time to analyse the reports and to plan the actions to be taken as a result. The more complex the game is, the more time the evaluation would take, as the agents would need more time to run and finish the play-through to provide feedback. A feasible solution would be presenting the game split into stages or levels; analysing small chunks each time. Also, the complexity of the game affects the performance of the algorithms as General AI has some limitations in solving complicated environments.

The methodology presented here would obviously be strengthened if these limitations would not exist or should they be minimised. Thus, it is also an aim of this paper to motivate and encourage research on these areas:

A. Reinforcement Learning

One of the limitations when working with *Reinforcement Learning* (RL) algorithms, is that they need off-line training and their performance depends on the size and intricacy of the system. They must explore the environment, having to decide between exploitation and exploration as it learns which actions lead to rewards [32]. The more complex the game is, the more they struggle as more the rewards are delayed in time.

There have been clear advances in RL methods, showing good performance in well-defined problems. An example is *AlphaGo* mastering *Go* [33]. Although the rules of the game are simple, it has certain characteristics that impeded AI to master it for a long time: deep games, large branching factor and, above all, lack of a good state evaluation function. Other examples showing the progress in RL, applied to video games, come from the work done by Mnih. *et al.* [6] (see Section II) and the research on the *VizDoom* platform [34].

Despite this progress, RL has not yet provided world winning approaches for more complex games, such as *Starcraft* [35]. Not only games like this require multiple levels of abstraction and reasoning but also include many real-world features that limit the application of these techniques. Examples are, in this and other games, the presence of a continuous state and action space, stochasticity, partial observability (fog of war is present in multiple strategy games) and multi-agent systems.

B. Planning Algorithms

These algorithms do not require off-line training (setting aside the parameters optimisation discussed in the next section) and therefore have a quick set-up. However, they require a forward model to be able to simulate possible future states to choose the best action available. Hence, there exists the challenge of having to create a forward model from scratch to include it in the game or working with abstract or not precise forward models available.

Moreover, the number of roll-outs (that depends on processing time and resources) have a big impact on the behaviour and performance, as the more simulations they are allowed to see, more information they get about the future. In [36], the authors compare the differences in performance in the Physical Travelling Salesman Problem when a budget of 40ms or 80ms is provided to MCTS, RHEA and RS. It has an impact on the number of roll-outs available per turn for the MCTS and the number of individuals for the Genetic Algorithms (GA). In the General AI scope, M. Nelson [37] ran a series of experiments through 62 games that form the GVGAI framework. The goal was checking how the performance of the MCTS is affected when varying the time budget provided to return an action, which influences the number of roll-outs it gets to take.

C. Parameter Optimisation

General AI algorithms use a series of parameters that have a big impact on their performance and behaviour and, in most of the cases, need to be optimised. As mentioned in the previous section, if the number of roll-outs available to the planning algorithm is modified, the number of predictions will be reduced or extended and, therefore, the information available to take a decision will be affected, influencing the results. In evolutionary algorithms, the size of the population has an impact on the performance. Gaina *et al.* compared how the winning rate of the RHEA was influenced by the size of the population and individual length [15], so the optimisation of the parameters is important indeed.

Optimising the parameters to the game under evaluation could take time. If not enough time is allowed, their expected performance could drop, which would end up providing misleading reports. The optimisation has usually been done off-line to provide enough time to reach a certain level of performance. However, there has been some recent progress, and an online adaptive parameter tuning mechanism for MCTS has been implemented in GVGP, with promising results [38].

The N-Tuple Bandit Evolutionary Algorithm (NTBEA) shows ways to mitigate some of the limitations presented by this parameter optimisation. Lucas *et al.* describe the NTBEA as a simple, informative and efficient model capable of being applied to numerous optimisation-related problems [39]. In the referenced work they show how to apply this approach to optimise the parameters of RHEA. In [40] Kunanusont *et al.* use this algorithm to evolve the game parameters of *Space Battle*, affecting its design. They argue how the results obtained in the experiments carried out show how NTBEA could be used for AI-Assisted Game Design.

The *team* should be well-tuned to allow the agents to recognise and carry out the actions expected to reach their goals, in order to obtain proper results that fit the expectations and interpret the feedback accordingly.

D. The Challenge of General AI

Developing algorithms capable of working through different games is a challenging task as it is not possible to use any game-specific information to guide them. Because of the difficulties of the problem, several approaches have been created and are being investigated to tackle it. Thus, General AI is an ongoing research. Even considering the latest improvements, the results of the *GVGAI Competition*[3] show how it is still not good enough to generalise to every kind of game. Even when the agents perform well in some games, there are games with a very low percentage of success; and any algorithm manages to perform uniformly good through all of them.

Furthermore, general algorithms can be applied to several areas in games: from one player simple games to multi-player collaborative games, where they need to work together to achieve a common objective. The variety of the problems to tackle increases the complexity of the generalisation.

V. Conclusion

This paper proposes a new methodology using General AI for assisting game design and testing and explains its features. It presents a series of differentiated goals to be applied to the general agents to play the game in different ways. Having agents focusing on targets that go beyond simply winning the game leads to specialists with distinct gameplay styles to use to extract information. The two type of reports and logging system generated can help the designer to check if their game under evaluation fulfils the expectations. The information retrieved can be used to detect bugs, balance the game or tweak its parameters. Because of the independence of the rules given by the generality of the AI, this approach allows an early integration in a game under development without requiring major modifications when it is extended or modified.

This proposed methodology is rooted in previous work on the field of general game AI, automatic playtesting and AI-assisted game design. It takes into account the needs of the games industry for efficient and accurate game testing and highlights interesting areas of future research. Several extensions in the methodology are possible, as including agents with different levels of skill, players tackling multiple objectives or adding collaborative and social-oriented profiles that can fit multi-player games. Also, it can create reports considering the objectives of the different specialists at once, combining the results obtained to analyse the information of multiple agents outputs, study their correlations and provide a greater level of granularity.

We believe that using general algorithms is the next step for automatic game design and testing in order to provide a portability non-existent in the approaches followed to date.

[3] http://www.gvgai.net/

Furthermore, introducing the option to choose between several algorithms with differentiated behaviours and skills adds flexibility to adapt the methodology to the characteristics of the game under evaluation.

ACKNOWLEDGMENT

This work was funded by the EPSRC CDT in Intelligent Games and Game Intelligence (IGGI) EP/L015846/1.

REFERENCES

[1] J. Levine, C. Bates Congdon, M. Ebner, G. Kendall, S. M. Lucas, R. Miikkulainen, T. Schaul, and T. Thompson, "General Video Game Playing," 2013.

[2] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The Arcade Learning Environment: An Evaluation Platform for General Agents," *Journal of Artificial Intelligence Research*, vol. 47, pp. 253–279, 2013.

[3] D. Perez-Liebana, S. Samothrakis, J. Togelius, T. Schaul, S. M. Lucas, A. Couëtoux, J. Lee, C.-U. Lim, and T. Thompson, "The 2014 General Video Game Playing Competition," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 8, no. 3, pp. 229–243, 2016.

[4] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI Gym," *arXiv preprint arXiv:1606.01540*, 2016.

[5] M. Johnson, K. Hofmann, T. Hutton, and D. Bignell, "The Malmo Platform for Artificial Intelligence Experimentation." in *IJCAI*, 2016.

[6] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-Level Control Through Deep Reinforcement Learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[7] M. C. Machado, M. G. Bellemare, E. Talvitie, J. Veness, M. Hausknecht, and M. Bowling, "Revisiting the Arcade Learning Environment: Evaluation Protocols and Open Problems for General Agents," *Journal of Artificial Intelligence Research*, vol. 61, pp. 523–562, 2018.

[8] M. Ebner, J. Levine, S. M. Lucas, T. Schaul, T. Thompson, and J. Togelius, "Towards a Video Game Description Language," 2013.

[9] T. Schaul, "A Video Game Description Language for Model-Based or Interactive Learning," in *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*. IEEE, 2013, pp. 1–8.

[10] R. D. Gaina, D. Pérez-Liébana, and S. M. Lucas, "General Video Game for 2 Players: Framework and Competition," in *Computer Science and Electronic Engineering (CEEC), 2016 8th*. IEEE, 2016, pp. 186–191.

[11] A. Khalifa, D. Perez-Liebana, S. M. Lucas, and J. Togelius, "General Video Game Level Generation," in *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, 2016, pp. 253–259.

[12] A. Khalifa, M. C. Green, D. Perez-Liebana, and J. Togelius, "General Video Game Rule Generation," in *Computational Intelligence and Games (CIG), 2017 IEEE Conference on*. IEEE, 2017, pp. 170–177.

[13] D. Perez-Liebana, S. Samothrakis, J. Togelius, S. M. Lucas, and T. Schaul, "General Video Game AI: Competition, Challenges and Opportunities," in *Thirtieth AAAI Conference on Artificial Intelligence*, 2016, pp. 4335–4337.

[14] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A Survey of Monte Carlo Tree Search Methods," *IEEE Transactions on Computational Intelligence and AI in games*, vol. 4, no. 1, pp. 1–43, 2012.

[15] R. D. Gaina, J. Liu, S. M. Lucas, and D. Pérez-Liébana, "Analysis of Vanilla Rolling Horizon Evolution Parameters in General Video Game Playing," in *European Conference on the Applications of Evolutionary Computation*. Springer, 2017, pp. 418–434.

[16] T. Joppen, M. U. Moneke, N. Schröder, C. Wirth, and J. Fürnkranz, "Informed Hybrid Game Tree Search for General Video Game Playing," *IEEE Transactions on Games*, vol. 10, no. 1, pp. 78–90, 2018.

[17] D. Perez-Liebana, J. Liu, A. Khalifa, R. D. Gaina, J. Togelius, and S. M. Lucas, "General Video Game AI: a Multi-Track Framework for Evaluating Agents, Games and Content Generation Algorithms," *arXiv preprint arXiv:1802.10363*, 2018.

[18] S. Roohi, J. Takatalo, C. Guckelsberger, and P. Hämäläinen, "Review of Intrinsic Motivation in Simulation-based Game Testing," in *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. ACM, 2018, p. 347.

[19] C. Holmgård, M. C. Green, A. Liapis, and J. Togelius, "Automated Playtesting with Procedural Personas through MCTS with Evolved Heuristics," *arXiv preprint arXiv:1802.06881*, 2018.

[20] T. S. Nielsen, G. A. Barros, J. Togelius, and M. J. Nelson, "General Video Game Evaluation Using Relative Algorithm Performance Profiles," in *European Conference on the Applications of Evolutionary Computation*. Springer, 2015, pp. 369–380.

[21] T. Machado, A. Nealen, and J. Togelius, "CICERO: Computationally Intelligent Collaborative EnviROnment for Game and Level Design," in *3rd workshop on Computational Creativity and Games (CCGW) at the 8th International Conference on Computational Creativity (ICCC'17)*.

[22] ——, "SeekWhence a Retrospective Analysis Tool for General Game Design," in *Proceedings of the 12th International Conference on the Foundations of Digital Games*. ACM, 2017, p. 4.

[23] C. Guerrero-Romero, A. Louis, and D. Perez-Liebana, "Beyond Playing to Win: Diversifying Heuristics for GVGAI," in *Computational Intelligence and Games (CIG), 2017 IEEE Conference on*. IEEE, 2017, pp. 118–125.

[24] R. Bartle, "Hearts, Clubs, Diamonds, Spades: Players Who Suit MUDs," *Journal of MUD research*, vol. 1, no. 1, p. 19, 1996.

[25] N. Yee, "The Gamer Motivation Profile: What We Learned from 250,000 Gamers," in *Proceedings of the 2016 Annual Symposium on Computer-Human Interaction in Play*. ACM, 2016, pp. 2–2.

[26] M. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos, "Unifying Count-Based Exploration and Intrinsic Motivation," in *Advances in Neural Information Processing Systems*, 2016, pp. 1471–1479.

[27] M. J. Nelson, "Game Metrics Without Players: Strategies for Understanding Game Artifacts." in *Artificial Intelligence in the Game Design Process*, 2011.

[28] V. Volz, D. Ashlock, and S. Colton, "Gameplay Evaluation Measures," *Artificial and Computational Intelligence in Games: AI-Driven Game Design (Dagstuhl Seminar 17471)*, p. 122, 2018.

[29] A. E. Elo, *The Rating of Chessplayers, Past and Present*. Arco Pub., 1978.

[30] M. E. Glickman, "Example of the Glicko-2 System," *Boston University*, 2012.

[31] T. Minka, R. Cleven, and Y. Zaykov, "TrueSkill 2: An Improved Bayesian Skill Rating System," Tech. Rep., March 2018.

[32] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement Learning: A Survey," *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.

[33] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the Game of Go with Deep Neural Networks and Tree Search," *nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[34] M. Kempka, M. Wydmuch, G. Runc, J. Toczek, and W. Jaśkowski, "Vizdoom: A Doom-Based AI Research Platform for Visual Reinforcement Learning," in *Computational Intelligence and Games (CIG), 2016 IEEE Conference on*. IEEE, 2016, pp. 1–8.

[35] S. Ontanón, G. Synnaeve, A. Uriarte, F. Richoux, D. Churchill, and M. Preuss, "A Survey of Real-Time Strategy Game AI Research and Competition in Starcraft," *IEEE Transactions on Computational Intelligence and AI in games*, vol. 5, no. 4, pp. 293–311, 2013.

[36] D. Perez, S. Samothrakis, S. Lucas, and P. Rohlfshagen, "Rolling Horizon Evolution Versus Tree Search for Navigation in Single-Player Real-Time Games," in *Proceedings of the 15th annual conference on Genetic and evolutionary computation*. ACM, 2013, pp. 351–358.

[37] M. J. Nelson, "Investigating Vanilla MCTS Scaling on the GVG-AI Game Corpus," in *Computational Intelligence and Games (CIG), 2016 IEEE Conference on*. IEEE, 2016, pp. 1–7.

[38] C. F. Sironi, J. Liu, D. Perez-Liebana, R. D. Gaina, I. Bravi, S. M. Lucas, and M. H. Winands, "Self-Adaptive MCTS for General Video Game Playing," in *International Conference on the Applications of Evolutionary Computation*. Springer, 2018, pp. 358–375.

[39] S. M. Lucas, J. Liu, and D. Perez-Liebana, "The N-Tuple Bandit Evolutionary Algorithm for Game Agent Optimisation," *arXiv preprint arXiv:1802.05991*, 2018.

[40] K. Kunanusont, R. D. Gaina, J. Liu, D. Perez-Liebana, and S. M. Lucas, "The N-Tuple Bandit Evolutionary Algorithm for Automatic Game Improvement," in *Evolutionary Computation (CEC), 2017 IEEE Congress on*. IEEE, 2017, pp. 2201–2208.

A Social Science-based Approach to Explanations for (Game) AI

Vanessa Volz, Kevin Majchrzak
TU Dortmund University
Department of Computer Science
vanessa.volz@cs.uni-dortmund.de, kevin.majchrzak@tu-dortmund.de

Mike Preuss
ERCIS, WWU Münster
Department of Information Systems
mike.preuss@uni-muenster.de

***Abstract*—The current AI revolution provides us with many new, but often very complex algorithmic systems. This complexity does not only limit understanding, but also acceptance of e.g. deep learning methods. In recent years, explainable AI (XAI) has been proposed as a remedy. However, this research is rarely supported by publications on explanations from social sciences. We suggest a bottom-up approach to explanations for (game) AI, by starting from a baseline definition of understandability informed by the concept of limited human working memory. We detail our approach and demonstrate its application to two games from the GVGAI framework. Finally, we discuss our vision of how additional concepts from social sciences can be integrated into our proposed approach and how the results can be generalised.**

***Index Terms*—explainable AI, working memory, super-sensors**

I. Introduction

The dream of intelligent machines has inspired research for decades. Currently, intelligent agents are starting to surpass human level performance in many areas that where originally dominated by humans. One of the most prominent examples is image recognition, where machines achieved lower error rates compared to human experts [1], [2].

Unfortunately, this trend does not only reward us with higher performance, but also entails increased complexity, and in many cases lower understandability of models used for artificial intelligence applications. All of the currently known best performing methods for image recognition, for example, are based on deep convolutional neural networks, a special form of artificial neural networks [2]. These models tend to be extremely complex and it seems to be virtually impossible for humans to understand them directly.

The lack of understandability of these models turns out to be a huge problem for many practical applications. This is especially true whenever models make decisions involving humans, as envisioned in many medical applications. It is thus very important to understand the decisions suggested by medical expert systems in order to avoid taking risks that could jeopardize a patient's health. This is the reason why non-transparent models are seldom applied in medical areas [3].

The *General Data Protection Regulation (EU)* became effective in May 2018 and further complicates this problem. It grants all EU citizens the right to receive explanations for all algorithmic decisions that have been made concerning them [4]. This will render black-box models useless for many applications if we fail to develop appropriate and understandable explanations.

Generally, it is a widespread claim that users do not trust algorithmic systems [5], [6] if they do not understand how they work. Incidentally, studies suggest that the public would be much more willing to accept algorithmic decision support if they did [7], [8], [9]. Additionally, one expects that the debugging [10] and also improvement of AI methods would be easier for explainable models.

As a result, the topic of *eXplainable AI* (XAI) resurged and regained traction in 2016, when the United States Defence Advanced Research Projects Agency (DARPA) started a funding line for XAI[1]. However, there is a lack of approaches that incorporate findings from social sciences [11] instead of the author's notion of understandability. With this paper, we therefore follow a bottom-up approach to generating explanations for AIs, based on the social sciences concept of working memory. We also demonstrate and envision how further concepts suggested in [12] can be applied to a practical example. To the best of our knowledge, this has not been done before.

For our proof of concept, we choose to focus on games. This is because games are designed for human players and should thus generally be understandable for humans. Additionally, XAI is also an important concern in the context of games, specifically in deceptive games [13] where short- and long-term rewards are conflicting [14]. Game designers should have at least a basic understanding of the behaviour of an AI in a game in order to be able to predict the agent's actions in untested environments, such as new levels. This is essential for planning the game and identifying unintended behaviour, but also for fine-tuning a given AI. From the perspective of the player, understanding AI behaviour is necessary for immersion, as well as for efficient collaboration with AI players.

In section II, we provide background information on *explanations* and *understandability*, the concept of *working memory*, as well as a survey of related work on explainable (game) AI. In section III, we describe our proposed bottom-up approach to generating explanations for the behaviour

[1] http://www.darpa.mil/program/explainable-artificial-intelligence

978-1-5386-4359-4/18/$31.00 ©2018 IEEE

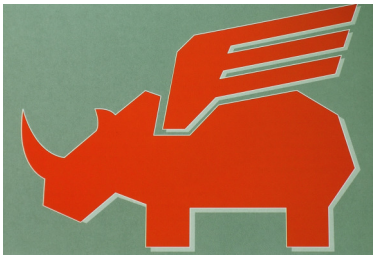


Fig. 1. Dortmund's flying rhinoceros by Josef Lehmkuhl. To interpret a model you do not need to know if it is correct or where it came from.

of (game) AIs based on the assumption of limited working memory. We provide a proof-of-concept application of our approach in section IV. Finally, in section V, we discuss our findings, and describe our vision for incorporating more social science concepts into XAI research.

II. Background

A. On Explanations and Understandability

First, we provide working definitions for the two terms explanation and understandability.

> To explain an event is to provide some information about its causal history. In an act of explaining, someone who is in possession of some information about the causal history of some event – explanatory information, I shall call it – tries to convey it to someone else. [15, p. 217]

Naturally, a successful explanation needs to be understood by the recipient.

> Someone can be said to understand a formal construct if he can relate it to the corresponding real-world entities and propositions and reason about the implications. [16, p. 12]

It is important to distinguish between the understandability of a model and its correctness or applicability. This is nicely illustrated by Rüping in [16] with the following example. The claim *"Rhinoceroses can fly."* is incorrect. However, it is generally understandable according to the above definition, as one could imagine winged rhinoceroses (figure 1) and reason about the resulting implications, e.g. for air safety.

Given this definition, the understandability of an explanation is still subjective. It depends on prior knowledge as well as the information processing and deduction capabilities of the human recipient on the one hand, but also on the presentation of the information on the other. Both aspects have been investigated in research fields such as philosophy, cognitive psychology/science, and social psychology. [12] gives a detailed overview of the current state-of-the-art as well as their potential applications to explanations of AI agents.

B. Capabilities and Limitations of the Human Mind

In order to understand an explanation, the information conveyed needs to be processed and stored for a short period of time. This process is limited by the *working memory* of the human brain, i.e. the capacity for "short-term storage and manipulation of sensory information lasting on the order of seconds" [17]. It is widely believed that working memory is limited, but the exact nature of these limitations are still debated [18].

The classical model of working memory is that it is of limited capacity, i.e. it holds a fixed, small number (K) of items, as famously suggested by G.A. Miller in [19]. This modelling approach is called a *slot model*, where every item is either recalled fully or not at all [18]. Newer approaches, so called *resource models*, understand working memory as a limited resource that is distributed among the information entities that need to be stored. According to these models, the higher the amount of resource allocated to some information, the more precisely it can be recalled [18]. In this paper, we will however be working with the slot model, as it is commonly used as a summary statistic [18] and easily applicable to our use case.

In his meta-study, G.A. Miller analyses the capability of the human mind to process information from an information theoretical perspective [19]. The presented results indicate that an average human brain is able to distinguish between 3 and 15 values for a one-dimensional variable. A one-dimensional variable can be, for example, the loudness of a tone or the intensity of a color. The tested participants were, for example, able to correctly match tones to up to 5 classes (such as very low, low, medium, high and very high). From 6 classes upwards they started to make mistakes and the amount of information that they were able to process quickly reached a limit. Interestingly, Miller observed this kind of limit on the human mind for a wide range of stimuli without exceptions. Furthermore, the observed limits turned out to be surprisingly low and very rarely higher than 15 classes.

According to the study, multi-dimensional information (such as faces) can be distinguished more easily, as the brain focuses on distinctive characteristics instead of storing all information. These observations match the subjective experience that multidimensional representations of variables, like colored plots and diagrams, are easier to understand than one-dimensional representations. However, the dimensionality can not be increased indefinitely, as the study finds that a human brain can only handle between 5 and 9 cognitive entities at the same time [19]. Thus, humans are strongly limited in their ability to observe multiple variables in relation to each other.

C. Related Work

Generating explanations for complex models to increase their transparency is an important topic in various fields of research, such as business process modelling [20] and object-oriented software [21]. (Deep) Neural networks are very prominent targets for explanation, with research published on feature visualisation, attribution, dimensionality reduction and rich user interfaces (see [22] for a survey and study on combining multiple approaches). An overview of general approaches to explanation and justification can be found in [23]. See also [2] for a recent overview over the current state

[2]https://medium.com/@jschwiep/the-state-of-explainable-ai-e252207dc46b

of XAI. According to [23], explanation approaches can be categorised as follows:

1) Inherently interpretable models: The agent makes decisions according to an inherently explainable model, such as rule-based systems or decision trees. However, there is still no consensus on whether these models are in fact generally interpretable [24].
2) Model-specific approaches: Model-specific characteristics are utilised to explain behaviour. Typical examples are methods that identify prominent composite features in neural networks as well as their attribution to the outcome [22].
3) Model-agnostic approaches: The agent is treated as a black-box. Approaches in this category can thus be applied independently of the agent or usecase [25], [26].

As the objective of this paper is to explain the behaviour of a given video game AI agent, only model-agnostic techniques are suitable. With our overview of related work, we thus focus on model-agnostic approaches to explanations.

While there is a considerable interest in explainable AI[3], studies confirm that published research is rarely informed from knowledge acquired in fields like social and behavioural sciences [11]. In fact, 11 out of 17 publications deemed important by the XAI community did not even include any references to papers on explanation from social sciences according to a study [11]. Most explanation approaches focus on identifying characteristic behaviour or instances, and require a human to interpret these examples and draw conclusions [27], [28]. Other approaches, such as [29], use strictly data-driven methods to imitate explanations by human experts.

While most of these publications are verified using surveys, these approaches can not guarantee or formally measure whether the resulting explanations are understandable. Based on these observations, in [12], the author calls for more interdisciplinary collaboration and provides a survey of suitable work from social science along with potential applications in AI. However, due to the generality of the work, the recommendations in [12] stay very abstract. By providing a specific use case, we hope to be able to demonstrate one such interdisciplinary approach as a first step in this direction.

In [16], Rping suggests a concept of interpretability that, like ours, is based on the famous study on working memory by Miller [19]. However, in contrast to our work, the proposed concept does not include Miller's findings on the capability of a human brain to distinguish classes. Additionally, [16] addresses classification models exclusively and specifically support vector machines, which of course have different properties than general AIs. We also provide additional insights on how other concepts from social sciences can be applied on top of the heuristics based on Miller's study.

A commonly observed weakness of explanations is, that they are only provided for a very narrow context of the application [26]. It is thus impossible to generate explanations on different levels of abstractions. In comparison, by relying on indicators defined by a human, our approach is suitable for multiple levels of abstraction.

[3]http://home.earthlink.net/~dwaha/research/meetings/ijcai17-xai/

III. AI Explanations Limited by Working memory

In the following, we propose an approach that incorporates insights on explanations from social sciences into a generally applicable algorithm to generate explanations of black-box AI agents. The explanations are intended to convey why a specific action was chosen over other available actions. The algorithm is informed only by intentional and observable actions [30] from the perspective of a single agent. We demonstrate that the explanations are understandable according to our definition based on the concepts on working memory described in section II-B.

A. Heuristics for Understandability

In [12], the author finds that explanations are selected, i.e. the information taken into account by the human recipient is a subset of the complete causal chain. Empirical studies also agree that simpler explanations are considered to be better, sometimes even regardless of their accuracy [31]. While there are many ideas on how exactly the explanations are selected to improve understandability, we first focus on the amount of information humans are able to process to provide a baseline for understandability.

Based on the study from Miller detailed in section II-B, we derive the following four heuristic rules for understandable models.

Definition 1 (Heuristics for understandable models): The following heuristics can be applied to all variables contained inside a given explanatory model. This includes inputs, outputs and intermediate results.

1) All variables should be understandable. This means that they need to have clear semantics the user can interpret.
2) One-dimensional variables should not take more than 3 to 15 values.
3) Multidimensional variables should also not take considerably more than 3 to 15 values.
4) The number of variables observed at the same time should never exceed 5 to 9.

The first heuristic rule takes into account the individual user's prior knowledge and level of abstraction. The second and third rules are based on the observed limits on the human mind to process information. The fourth rule takes into account the discussed limits on the human working memory. The last three heuristic rules are also closely related to what is often casually referred to as "model complexity".

We want to stress that these heuristics are aimed at necessary conditions for understandability, because they limit the information that needs to be processed according to the findings by Miller [19]. Additional concepts from social sciences should also be incorporated in order to express sufficient conditions for understandability, as discussed in section V. Furthermore, these heuristics are based on [19], but they should nevertheless be validated with a survey, especially

given that the model proposed in [19] has been called into question [18].

Given new findings, these heuristics can of course be adapted accordingly. For the purposes of this paper, we will use them in order to identify how they can be enforced on models used to generate explanations for AIs.

B. Super-Sensors

According to our first heuristic, all variables in the explanation should be understandable. To ensure this, we use super-sensors, which are mappings from raw sensory data to understandable concepts. Similar ideas have been proposed in [26], [32]. Another related idea is the identification of understandable features, such as an ear in context of explaining neural networks for picture classification [22].

Definition 2 (super-sensor): More formally, let $\mathcal{I}$ denote a set of possible inputs. A **super-sensor** for $\mathcal{I}$ is a function $s\colon \mathcal{I} \mapsto \mathbb{B}$ with $\mathbb{B} = \{0, 1\}$.

Super-sensors map the original inputs to values $b \in \{0, 1\}$. They represent the presence or absence of features in the original inputs. We have chosen a binary mapping here, as it is clearly within the limitations defined by our understandability heuristics (definition 1). However, mappings could be extended to allow for multiple classes. Even using fuzzy logic to express something like proximity is conceivable.

The explanation is framed in terms of the super-sensors. This way, they can also be used to adjust to the desired level of abstraction, the specific question posed or to the prior knowledge of the human user. As a consequence, this step has the potential to induce implicit bias in the explanation. Additionally, a given set of super-sensors might not be able to capture all observed agent behaviors. They should, thus, be defined carefully and based on a given use case. In the following, we list some research that may be helpful in this regard.

The simulation heuristic [33], for example, can be used to develop several guidelines for the selection of super-sensors. The hypothesis they propose is that humans, when asked about past events, simulate alternative cases in order to improve their own understanding. According to this theory, explanations are framed based on the facts that are mutated in these simulations, similarly to how super-sensors frame our explanation approach. Research has found that the mutated facts are recent rather than distal [34]. Additionally, facts that are deemed more controllable by the actor are mutated more frequently [35]. Super-sensors should, thus, describe current or more recent states as well as states that can be actively influenced by the AI agent.

Furthermore, explanations are categorised as either functional or mechanistic. Functional explanations take into account the purpose or function of a behaviour, while mechanistic explanations address more the circumstances and mechanics that cause the behaviour [36]. Functional explanations are of course very important in the context of explainable AI. However, research has also discovered that mechanistic explanations are often preferred when explaining failed behaviours [37]. For this reason, super-sensors should on the one hand include information that is available to the AI agent and thus likely used for its reasoning (functional). But on the other hand, some sensors should also contain information on events and obstacles that could hinder the execution of a strategy, such as e.g. enemies (mechanistic).

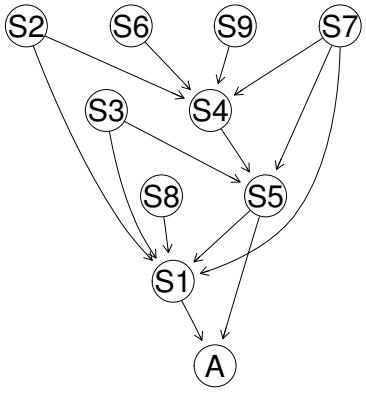


Fig. 2. Example for a Bayesian network model as described in subsection III-C. The node A without outgoing edges represents the agent's actions. The other nodes represent the super-sensors used to predict the agent's actions.

C. A Bayesian network model

In this paper, we use Bayesian network models (see e.g. [38] for detailed description) to generate explanations. Bayesian networks are probabilistic graphical models represented by directed acyclic graphs. The graph's nodes represent random variables and the directed edges between them represent correlations between the variables. Parents and children are modelled to be in a causal relationship where the parent's value influences the child's value.

There exists extensive work on the understandability of Bayesian networks, as surveyed in [39]. Additionally, because Bayesian networks are probabilistic models, they identify explanations that are true or likely, which is one objective of explanations [40].

In order to use Bayesian networks for our purposes, we create a random variable S_i for each super-sensor s_i and another random variable A for the agent's possible actions. These random variables represent the nodes of the Bayesian network to be trained. Figure 2 shows an example for such a network. The random variables for the super-sensors take on the values *true* or *false* depending on whether the super-sensor values are 1 or 0. The domain of the random variable A is the set of possible actions $\mathcal{A}$.

Structure and parameters of the Bayesian network can be learned based on pairs of super-sensor values and actions. Super-sensor values, in turn, can be calculated based on game states. Therefore, we can generate training data for the Bayesian network simply by observing an agent's actions along with the gamestate while playing the game.

However, we need to ensure that the trained Bayesian network is understandable according to our heuristics (definition

1). To do this, we first restrict the structure of the network in such a way that only incoming edges are allowed for action node A. This a legitimate application of domain knowledge, as we want to investigate the causal history of an action for a given gamestate.

Then, we utilise the property of Bayesian networks that each node is conditionally independent of its non-descendants given its parents. In figure 2, given S1 and S5, A is conditionally independent of all other sensor values. Because of our imposed structural restriction, node A will never have any descendants and the probability distribution can thus be fully expressed given its parent nodes. If we now limit the number of parent nodes for A (to 7 as suggested in [19]), we ensure adherence to heuristic 4.

IV. Proof of Concept

In the following, we demonstrate how the approach described in the previous section can be applied to a specific use case. As a practical example, we have chosen computer games and agents from the *General Video Game AI (GVGAI)* competition [41]. More specifically, two games involving vertically traversing projectiles. The first is the game *Aliens* similar to the Atari classic *Space Invaders*. The second one is *Eggomania*, where a chicken throws eggs which the player needs to catch before they hit the ground.

In each time step, the agent can observe the latest game state and has access to a forward model to simulate possible future game states. Based on this information, it then chooses one of the 6 possible actions $\mathcal{A} = \{$ACTION_LEFT, ACTION_RIGHT, ACTION_UP, ACTION_DOWN, ACTION_USE, ACTION_NIL$\}$. The first four actions are used to navigate the avatar. ACTION_USE can be used to let the avatar interact with its environment and ACTION_NIL indicates no action. As only 6 actions are possible, no further abstraction is required in order to fulfil heuristic 2.

A. Super-Sensors

We manually defined 19 super-sensors for the game Aliens and 14 super-sensors for Eggomania. They represent features with plain semantics that we assumed to be relevant to an agent's behaviour. One of the super-sensors for the game Aliens, for example, is true *iff* an enemy is in range and can be shot. Another one of the super-sensors for Aliens is true *iff* the player's avatar is about to get hit by an enemy projectile.

To adhere to the guidelines discussed in section III-B, all super-sensors correspond to current events or ones in the near future. The sensors also exclusively express states that can be influenced by the agent by either moving or shooting.

B. Training the model

We used 100 play traces per agent and game for training. The structure and parameters of the Bayesian networks have been learned via the R package *bnlearn* [42]. We employed the improved hill climbing algorithm [38], [42] for structure learning and *Bayesian parameter estimation* [38] for parameter learning. For inference in the learned networks we used the R package *grain* [43] because it allows exact inference in Bayesian networks.

TABLE I
Win rates for the original agent, maxAgent and stochAgent.

testcase	Agent	maxAgent	stochAgent
Aliens-Agent	100%	99,5%	98,5%
Eggomania-Agent	100%	100%	100%
MaastCTS2 Aliens	30,66%	0%	0%
MaastCTS2 Eggomania	97,66%	12,33%	97,17%
YOLOBOT Aliens	99,83%	0%	6%
YOLOBOT Eggomania	99,66%	85%	20,5%

C. Evaluation Approach

We evaluate the generated explanations regarding the three following aspects:

1) How accurately are causal structures captured?
2) Does the explanation generalise to other situations?
3) Which causal structures are detected?

All evaluations are based on 500 play traces per agent and game.

In order to assess the degree to which agent behaviour is explained (aspect 1), we assess the accuracy of the representation by the Bayesian network. To do this, we measure how accurately our explanation model predicts the agent's actions given super-sensor input. In this context, the prediction accuracy is called *action agreement ratio (AAR)* [44]. It is the ratio between the number of correctly predicted actions and the number of all actions. The AAR, therefore, takes on the value 1 if the agent's actions have been predicted perfectly.

However, a perfect AAR often is not achievable. One potential issue is the suitability of the specification of super-sensors. E.g, some crucial information might be missing in the abstract representation of the gamestate via the super-sensors. Additionally, agent behaviour is not necessarily deterministic, causing lower action agreement because the Bayesian network model is. In order to assess the gravity of the latter, we also compute the *best possible AAR*, i.e. the AAR achieved by the best possible hypothetical model, given the super-sensors, that always predicts the action that is taken in the majority of observations. This value is then used in order to normalise the AAR. The best possible AAR thus also provides means to measure the quality of the super-sensor set.

If the explanations are able to capture an abstraction of agent behaviour, this should be generalisable to different situations in both games (aspect 2). We thus test the in-game performance of agents that use the Bayesian network as a decision guideline. The observed win rates are then compared with the results of the original agents. We use two types of agents: *maxAgent*, which for any given state will always choose the action with the highest probability according to the Bayesian network, and *stochAgent*, which will randomly choose an action from the probability distribution encoded in the network.

Finally, we also address which of the super-sensors are identified as parent nodes in the Bayesian network (aspect 3).

D. Results

We first conduct an experiment in order to find out if our approach is generally able to model agent behaviour, while adhering to the understandability heuristics defined. To this end, we implement deterministic agents for both games (*Aliens-Agent* and *Eggomania-Agent*, respectively), that choose their actions based on subsets of the implemented super-sensors. Therefore, it should be possible to learn the correct explanation model for these agents and to predict their actions perfectly based on the super-sensors. In our tests, the Bayesian networks indeed achieved almost perfect action agreement ratios. The win rates of the Bayesian network-based and original agents are also very similar (cf. table I).

We further compare the super-sensors in the Bayesian network with the ones we chose to implement the *Aliens-Agent* and *Eggomania-Agent*. We find that either the same or equivalent sensors were selected. Additionally, the set of super-sensors also contains a sensor that randomly returns values as a sanity check that only sensible sensors are selected. We confirmed that none of the trained networks picked the random sensor. With these results, we conclude that the approach can produce sensible explanations where we know appropriate solutions exist.

Next, we evaluate our approach for two arbitrary black-box agents for both games. We elected to choose one agent that relies on unknown heuristics, and another one that does not contain obvious domain knowledge. For the heuristics-agent, we modified *MaastCTS2* [45], the winning agent of the GECCO GVGAI competition 2016, such that only its heuristics are used without the Monte Carlo tree search (MCTS) component. The second agent was *YOLOBOT* [46], the winner of the CIG GVGAI competition 2016. This agent uses MCTS, as well as a range of heuristics and a knowledge base that is built during game play to decide on its actions. The achieved, best possible and normalised AARs for all combinations of agents and games are visualised in figure 3.

For the game Aliens, we reach a normalised mean AAR of above 0.7 and for Eggomania, of above 0.9. That means that the Bayesian networks are successful in explaining the data. However, except for MaastCTS2 and Eggomania, the best possible AAR values are relatively low (between 0.4 and 0.5). This indicates that the chosen super-sensor set was not sufficient to fully inform on the behaviour. However, the identification of super-sensors was not the main component of this paper, but should be investigated in future work.

A similar conclusion can be drawn from the results of the win rates (table I). It seems that if the super-sensors are defined appropriately, the explanation model is able to accurately imitate the original agent and produce similar win rates. In this case, the explanation generated also aligns with the interpretations of playthroughs by a human observer (e.g., in Aliens, sensors related to bomb and shield locations are chosen).

V. Vision

A. Summary and Conclusions

In this paper, we have investigated explanations for AI behaviour that adhere to limitations of working memory according to research in social sciences [19]. We have developed a general approach to generating explanations that respect these limitations and demonstrated its application for two games. The proof-of-concept shows that the proposed approach is generally suitable. However, there are still several potential improvements.

For example, as discussed in section II-B, the slot model for working memory has been challenged recently. It therefore should be investigated how models can be created that take memory limits into consideration following a resource model instead. Furthermore, ideally, understandable super-sensors adhering to the concepts described in III-B could be identified automatically, in order to make the approach more easily generalisable. Additionally, the measure used in our example to express explanatory accuracy is very limited. The action agreement ratio can only assess explanations on a very low level, i.e. regarding actions. However, in many cases, more interesting questions would address the intentions and goals of the AI [12]. Finally, besides measures to assess accuracy, the method should also be evaluated in terms of its explanatory power, e.g. via a survey or using concepts from related research in social sciences. Some of these improvements are discussed in detail in the following.

B. Beyond Causal Attribution

Bayesian network models are very suitable in order to attribute observable causes to recorded outcomes. However, beyond strict causal attribution, social sciences also investigate how humans explain behaviours. One of the most important concepts in this regard is intentionality [12]. The main distinctive factor between intentional and non-intentional actions is that the former are *equifinal* [47], i.e. if an actor fails to achieve a certain state in a certain way, it will attempt to find another one. This characteristic could be utilised in order to identify intentions automatically, in order to enable explanations on a more abstract level. The importance of the respective goals should correspond to how consistently they are pursued. Measuring this persistence, however, requires measures for partial fulfillment of goals as well as correlations between different goals.

Alternatively, explanations could of course also be improved by information on how decisions are made is obtained from the acting AI agent. This would reduce the information (actor-observer) asymmetry [48]. Information that would be specifically suitable in this regard are the fitness functions popular search-based algorithms (such as MCTS, EAs) use. Further AI-specific information, e.g. on decisiveness, would also be useful.

Furthermore, in this paper, we have restricted the amount of information that needs to be processed simultaneously by configuring a Bayesian network accordingly. As a result, the

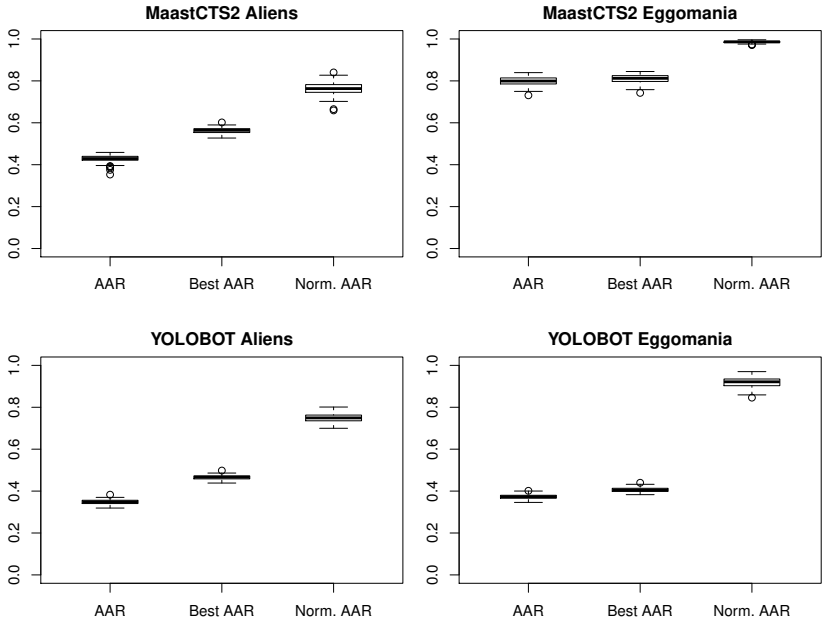


Fig. 3. Achieved, best possible and normalised AAR for MaastCTS2 and YOLOBOT on Aliens and Eggomania

super-sensors that explain most of the observed behaviour will be selected. However, while this is a positive property for an explanation [40], explanatory power is also influenced by other properties, which are discussed in the following.

One aspect is that humans often restrict their explanations to abnormal behaviour [49], [40], [50]. In this case, it is assumed that all parties involved have some common understanding of a norm. The implicit foil to a question on reasons for unexpected actions is thus the behaviour considered to be normal. These insights could be used to specifically target AI behaviour that differs from human agents. Given suitable playtraces from human and AI players, unexpected behaviour could be identified based on the differences between the observed actions following all possible combinations of super-sensors. This information could then be used to explicitly explain unexpected behaviour, by training the Bayesian network only on abnormal observations. Alternatively, nodes in a Bayesian network corresponding to super-sensor combinations related to unexpected behaviour could be highlighted to improve understandability.

There are several other characteristics, that could be incorporated into the training process as detailed above. For example, humans also prefer highly contrastive explanations [50], [51], i.e. instances, where the action taken and its foil are very distinctive. These actions could be identified using information criteria or by identifying super-sensors that produce very lopsided distributions of actions taken. Humans are also specifically interested in necessary causes [51]. In our example, super-sensors could be identified that are always true or always false when a specific action is taken. Especially in games, necessary causes are very meaningful, as they can explain multi-stage games like Pac-Man. A further interesting property is responsibility, i.e. how large the influence of a super-sensor is on the action taken by the agent [52]. An assessment of responsibility can be provided by computing how many super-sensor values need to be mutated to change the action chosen.

Another aspect of explanations, which has not been addressed in this paper, is that they are a form of conversation [53]. It is thus also important how they are communicated. In [29], the authors, for example, use natural language and imitate how human players communicate.

C. Beyond Aliens and Eggomania

In this paper, we test the application of the proposed approach using the games Aliens and Eggomania from the GVGAI framework. However, it is important to question whether it can also be generalised to other games. While we have given a general description in section III and the trained Bayesian networks will always be understandable according to the heuristics in definition 1, the main problem will be to define appropriate super-sensors. This was also demonstrated by our experimental results in section IV-D. As discussed in the previous section, further work on abstracting strategies from actions is needed here. The results could also be helpful when implementing intentional, believable and strategic AI.

REFERENCES

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[2] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.

[3] R. Caruana, Y. Lou, J. Gehrke, P. Koch, M. Sturm, and N. Elhadad, "Intelligible models for healthcare: Predicting pneumonia risk and hospital 30-day readmission," in *International Conference on Knowledge Discovery and Data Mining*, 2015, pp. 1721–1730.

[4] B. Goodman and S. Flaxman, "European Union regulations on algorithmic decision-making and a "right to explanation"," in *ICML Workshop on Human Interpretability in Machine Learning*, 2016, pp. 26–30.

[5] K. Stubbs, P. J. Hinds, and D. Wettergreen, "Autonomy and Common Ground in Human-Robot Interaction: A Field Study," *IEEE Intelligent Systems*, vol. 22, no. 2, pp. 42–50, 2007.

[6] M. P. Linegang, H. A. Stoner, M. J. Patterson, B. D. Seppelt, J. D. Hoffman, Z. B. Crittendon, and J. D. Lee, "Human-Automation Collaboration in Dynamic Mission Planning: A Challenge Requiring an Ecological Approach," *Human Factors and Ergonomics Society Annual Meeting*, vol. 50, no. 23, pp. 2482–2486, 2006.

[7] J. E. Mercado, M. A. Rupp, J. Y. C. Chen, M. J. Barnes, D. Barber, and K. Procci, "Intelligent Agent Transparency in HumanAgent Teaming for Multi-UxV Management," *Human Factors*, vol. 58, no. 3, pp. 401–415, 2016.

[8] J. Y. Chen, K. Procci, M. Boyce, J. Wright, A. Garcia, and M. Barnes, "Situation AwarenessBased Agent Transparency," Army Research Laboratory, Tech. Rep. ARL-TR-6905, April 2014.

[9] B. Hayes and J. A. Shah, "Improving Robot Controller Transparency Through Autonomous Policy Explanation," in *International Conference on Human-Robot Interaction*. ACM, 2017, pp. 303–312.

[10] T. Kulesza, M. Burnett, W.-K. Wong, and S. Stumpf, "Principles of Explanatory Debugging to Personalize Interactive Machine Learning," in *International Conference on Intelligent User Interfaces*. New York, NY, USA: ACM, 2015, pp. 126–137.

[11] T. Miller, P. Howe, and L. Sonenberg, "Explainable AI: Beware of inmates running the asylum," in *Proceedings of the IJCAI-17 Workshop on Explainable AI (XAI), Melbourne, Australien*, 2017, pp. 36–42.

[12] T. Miller, "Explanation in Artificial Intelligence: Insights from the Social Sciences," *CORR*, vol. abs/1706.07269, 2017. [Online]. Available: http://arxiv.org/abs/1706.07269

[13] D. Anderson, M. Stephenson, J. Togelius, C. Salge, J. Levine, and J. Renz, "Deceptive Games," in *Applications of Evolutionary Computation (EvoApplications '18)*, 2018, pp. 376–391.

[14] N. Companez and A. Aleti, "Can Monte-Carlo Tree Search learn to sacrifice?" *Journal of Heuristics*, vol. 22, no. 6, pp. 783–813, 2016.

[15] D. Lewis, "Causal Explanation," in *Philosophical Papers Vol. Ii*. Oxford University Press, 1986, pp. 214–240.

[16] S. Rüping, "Learning interpretable models," Ph.D. dissertation, TU Dortmund University, 2006.

[17] A. Baddeley, "Working memory: looking back and looking forward, journal = Nature Reviews Neuroscience, volume = 4, pages = 829–839, year = 2003, publisher = Nature Publishing Group."

[18] W. J. Ma, M. Husain, and P. M. Bays, "Changing concepts of working memory," *Nature neuroscience*, vol. 17, no. 3, pp. 347–356, 2014.

[19] G. A. Miller, "The magical number seven, plus or minus two: Some limits on our capacity for processing information," *The Psychological Review*, vol. 63, no. 2, pp. 81–97, 1956.

[20] H. A. Reijers and J. Mendling, "A Study Into the Factors That Influence the Understandability of Business Process Models," *Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans*, vol. 41, no. 3, pp. 449–462, 2011.

[21] M. Nazir and R. A. Khan, "An empirical validation of understandability quantification model," *Procedia Technology*, vol. 4, pp. 772–777, 2012.

[22] C. Olah, A. Satyanarayan, I. Johnson, S. Carter, L. Schubert, K. Ye, and A. Mordvintsev, "The Building Blocks of Interpretability," *Distill*, 2018, https://distill.pub/2018/building-blocks.

[23] O. Biran and C. Cotton, "Explanation and justification in machine learning: A survey," in *IJCAI-17 Workshop on Explainable AI (XAI)*, 2017, pp. 8–13.

[24] Z. C. Lipton, "The Mythos of Model Interpretability," in *ICML Workshop on Human Interpretability in Machine Learning*, 2016, pp. 96–100.

[25] M. T. Ribeiro, S. Singh, and C. Guestrin, "Model-Agnostic Interpretability of Machine Learning," in *ICML Workshop on Human Interpretability in Machine Learning*, 2016, pp. 96–100.

[26] ——, "Why should I trust you?: Explaining the predictions of any classifier," in *International Conference on Knowledge Discovery and Data Mining*, 2016, pp. 1135–1144.

[27] R. Turner, "A model explanation system," in *International Workshop on Machine Learning for Signal Processing*, 2016, pp. 1–6.

[28] P. Tamagnini, J. Krause, D. A., and B. E., "Interpreting black-box classifiers using instance-level visual explanations," in *Workshop on Human-In-the-Loop Data Analytics*, 2017.

[29] U. Ehsan, B. Harrison, L. Chan, and M. Riedl, "Rationalization: A Neural Machine Translation Approach to Generating Natural Language Explanations," in *Artificial Intelligence, Ethics, and Society*, 2018.

[30] B. F. Malle and G. E. Pearce, "Attention to Behavioral Events During Interaction: Two ActorObserver Gaps and Three Attempts to Close Them," *Journal of Personality and Social Psychology*, vol. 81, no. 2, pp. 278–294, 2001.

[31] T. Lombrozo, "Simplicity and probability in causal explanation," *Cognitive Psychology*, vol. 55, no. 3, pp. 232–257, 2007.

[32] S. Lundberg and S.-I. Lee, "An Unexpected Unity Among Methods for Interpreting Model Predictions," in *NIPS Workshop on Interpretable Machine Learning in Complex Systems*, 2016.

[33] D. Kahneman and A. Tversky, "The simulation heuristic," in *Judgment under Uncertainty: Heuristics and Biases*, D. Kahneman, P. Slovic, and A. Tversky, Eds. Cambridge University Press, 1982, pp. 201–208.

[34] D. T. Miller and S. Gunasegaram, "Temporal Order and the Perceived Mutability of Events: Implications for Blame Assignment," *Journal of Personality and Social Psychology*, vol. 59, no. 12, pp. 1111–1118, 1990.

[35] V. Girotto, P. Legrenzi, and A. Rizzo, "Event controllability in counterfactual thinking," *Acta Psychologica*, vol. 78, no. 1, pp. 111–133, 1991.

[36] T. Lombrozo, "Explanation and categorization: How "why?" informs "what?"," *Cognition*, vol. 110, no. 2, pp. 248–253, 2009.

[37] ——, "Causalexplanatory pluralism: How intentions, functions, and mechanisms influence causal ascriptions," *Cognitive Psychology*, vol. 61, no. 4, pp. 303–332, 2010.

[38] D. Koller and N. Friedman, *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009.

[39] C. Lacave and F. J. Díez, "A Review of Explanation Methods for Bayesian Networks," *Knowledge Eng. Review*, vol. 17, no. 2, pp. 107–127, 2002.

[40] D. J. Hilton, "Mental Models and Causal Explanation: Judgements of Probable Cause and Explanatory Relevance," *Thinking & Reasoning*, vol. 2, no. 4, pp. 273–308, 1996.

[41] D. Perez-Liebana, S. Samothrakis, J. Togelius, T. Schaul, S. M. Lucas, A. Couëtoux, J. Lee, C. Lim, and T. Thompson, "The 2014 General Video Game Playing Competition," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 8, no. 3, pp. 229–243, 2016.

[42] M. S., "Learning bayesian networks with the bnlearn R package," *Journal of Statistical Software*, vol. 35, no. 3, pp. 1–22, 2010.

[43] S. Højsgaard, "Graphical independence networks with the gRain package for R," *Journal of Statistical Software*, vol. 46, no. 10, pp. 1–26, 2012.

[44] C. Holmgård, A. Liapis, J. Togelius, and G. N. Yannakakis, "Personas versus Clones for Player Decision Modeling," in *Entertainment Computing*, 2014, pp. 159–166.

[45] D. J. N. J. Soemers, C. F. Sironi, T. Schuster, and M. H. M. Winands, "Enhancements for Real-Time Monte-Carlo Tree Search in General Video Game Playing," in *Conference on Computational Intelligence and Games*, 2016.

[46] T. Joppen, M. Moneke, N. Schroder, C. Wirth, and J. Furnkranz, "Informed Hybrid Game Tree Search for General Video Game Playing," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 10, no. 1, pp. 78–90, 2018.

[47] F. Heider, *The psychology of interpersonal relations*. New York: Wiley, 1958.

[48] B. F. Malle, J. Knobe, and S. E. D. Nelson, "Actor-observer asymmetries in explanations of behavior: new answers to an old question." *Journal of personality and social psychology*, vol. 93 4, pp. 491–514, 2007.

[49] D. Hilton and B. R. Slugoski, "Knowledge-Based Causal Attribution. The Abnormal Conditions Focus Model," *Psychological Review*, vol. 93, no. 1, pp. 75–88, 1986.

[50] G. Hesslow, "The Problem of Causal Selection," in *Contemporary Science and Natural Explanation: Commonsense Conceptions of Causality*, D. J. Hilton, Ed. New York University Press, 1988.

[51] P. Lipton, "Contrastive Explanation," *Royal Institute of Philosophy Supplement*, vol. 27, pp. 247–266, 1990.

[52] H. Chockler and J. Y. Halpern, "Responsibility and Blame: A Structural-Model Approach," *Journal of Artificial Intelligence Research*, vol. 22, pp. 93–115, 2004.

[53] D. J. Hilton, "Conversational processes and causal explanation," *Psychological Bulletin*, vol. 107, no. 1, pp. 65–81, 1990.

Towards Game-based Metrics for Computational Co-Creativity

Rodrigo Canaan
Tandon School of Engineering
New York University
Brooklyn, NY, USA
rodrigo.canaan@nyu.edu

Stefan Menzel
Honda Research Institute Europe GmbH
Offenbach/Main, Germany
stefan.menzel@honda-ri.de

Julian Togelius
Tandon School of Engineering
New York University
Brooklyn, NY, USA
julian.togelius@nyu.edu

Andy Nealen
Tandon School of Engineering
New York University
Brooklyn, NY, USA
nealen@nyu.edu

***Abstract*—We propose the following question: what game-like interactive system would provide a good environment for measuring the impact and success of a co-creative, cooperative agent? Creativity is often formulated in terms of novelty, value, surprise and interestingness. We review how these concepts are measured in current computational intelligence research and provide a mapping from modern electronic and tabletop games to open research problems in mixed-initiative systems and computational co-creativity. We propose application scenarios for future research, and a number of metrics under which the performance of cooperative agents in these environments will be evaluated.**

***Index Terms*—artificial intelligence, cooperative systems, games**

I. INTRODUCTION

Designing intelligent agents characterized by a co-creative, cooperative behavior would mark a major breakthrough in the age of industrial man-machine interaction. Exchanging relevant information with suitable time frequency and enriching the partner (human or machine) with novel perspectives and solution strategies on the problem are key factors for desirable results (considering the value of the output and the effort required). Cooperative games offer the valuable opportunity to realize an interactive environment for developing and evaluating computational methods used by these agents.

In this paper we review concepts and implementations of cooperative games in the light of their capability to impact development processes in (industrial) environments with co-evolution and co-creativity as important expressions for cooperation. Having a working definition of computational creativity, and how creative systems and their outputs are judged in terms of their value, novelty, interestingness, and surprise, will help us understand cooperatively creative agents and might help us build them as well. Computational creativity and AI-assisted design are important application areas for computational intelligence techniques such as neural networks, reinforcement learning and evolutionary computation; further, the conceptualization of creativity as search in a design space fits well with design applications of evolutionary computation.

Essentially, this paper tries to answer the following question: what game-like interactive system would provide a good environment for measuring the impact–and success– of a co-creative, fully cooperative agent?

We begin with a survey of the definition of computational creativity-related terms in the literature, how they relate to each other and how they apply to future work on our own co-creative agents in Section II. When considering cooperation between multiple actors (be they human or machine), in addition to the abilities and characteristics of each individual, the attributes of the relationships between individuals and the surrounding environment also impact the success of the endeavor. Section III explores some of these relational or environmental attributes of creative efforts, such as the exchange of information and the share of responsibility. In section IV we propose a set of metrics under which to evaluate cooperative agents in game-like environment, and section V gives our vision of how cooperative agents integrating all discussed techniques should operate in the long term.

II. COMPUTATIONAL CREATIVITY

Creativity is often understood as the production of novel and valuable concepts [1]. Computational creativity is a subfield of Artificial Intelligence (AI) that focuses on computational systems whose behavior can be deemed creative [2]. While much theoretical and practical work exists on systems that aim to be creative in their own right, with little or no human intervention [3]–[6], there are also many systems designed to cooperate with humans to achieve better results than either can presently do alone [7]–[9]. We focus on concepts of computational creativity and how they map to game-based tasks to further propose a number of concrete game-based metrics for co-creativity in a computational setting.

A. Novelty, Interestingness, Surprise

In his CSF framework [10], Wiggins says an artifact produced by a system is novel if there are no previously existing similar or identical artifacts in the context in which the artifact is produced. Ritchie [11] builds upon Wiggins' work and introduces the notion of the inspiring set as the "knowledge base of known examples which drives the computation within the program". Ritche calls an artifact generated by the program novel if it is not part of the inspiring set (or not too similar to its members). Both authors admit the possibilities of Novelty being either an absolute assessment (based on the existence of

978-1-5386-4359-4/18/$31.00 ©2018 IEEE

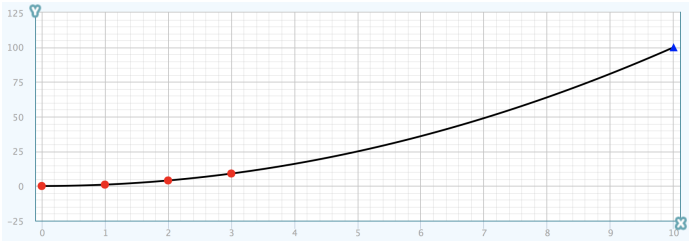


Fig. 1. A simple polynomial regression model trained on the dataset $I = [(0,0),(1,1),(2,4),(3,9)]$ (in red) would perfectly predict the point $x = (10,100)$ in blue, even though the Euclidean distance to the inspiring set is large

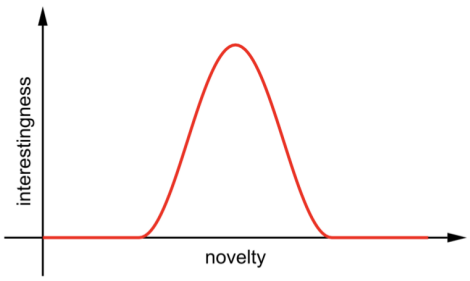


Fig. 2. A Wundt Curve, as shown in [13]

identical artifacts) or, more flexibly, to depend on some metric that establishes degrees of similarity between objects.

Reehuis *et al.* [12] provide an overview of Novelty metrics used by researchers, and propose dividing them between distance-based metrics and learning-based metrics. Distance-based metrics depend only on the distance, in a specified metric space, between a candidate solution and the archive of earlier solutions (what Ritchie would call the inspiring set). They define uniqueness as the minimum distance between a solution and a member of the archive, as used by [13] and [14]. Sparseness is defined as the average distance from a candidate solution to its k nearest neighbors in the archive, as used by Lehman and Stanley [15]. Reehuis *et al.* note that uniqueness is equivalent to sparseness with a value of k = 1.

Learning-based metrics take the agents expectations into account. Formally, let an agent (or an external viewer) be imbued with a model of the world, which ascribes probabilities to certain events. High novelty, or surprise occurs when the agent comes into contact with examples which contradict the model. Reehuis *et al.* provide the prediction error, dispersion in predictions and predictive variance of the model as examples of learning-based novelty. Itti and Baldi [16] provide a bayesian denition of surprise using the relative entropy, or Kullback-Leibler (KL) divergence [17]. Since the KL divergence depends on a prior probability distribution, we could also classify it as learning-based novelty.

B. Analysis of distance- and learning-based novelty metrics

We provide a simple example of the distinction between the two kinds of novelty in figure 1. The points in red are part of the inspiring set $I = [(0,0),(1,1),(2,4),(3,9)]$ and a candidate solution $x = (10,100)$ is shown in blue. A naive Euclidean distance-based metric would ascribe high novelty to x, while a simple learning model based on polynomial regression could might ascribe zero novelty to x, since it is a perfect fit to the parabola $y = x^2$. Thus, under learning-based novelty, what is novel to one observer might not be to another.

It is clear to us that the distinction between distance and learning-based novelty is didactic only. A high novelty value in a distance-based metric such as sparseness or uniqueness is equivalent to a low probability in a simple model that takes only the Euclidean distance from the points in the inspiring set into account (with more distant points being less probable). On the other hand, a more complex learning model can be abstracted as a distance metric in a sufficiently high dimension.

Thus, the choice of novelty metric to use depends on the problem. If one must describe a model being refined over time, or multiple agents with individual models making different predictions, a learning-based metric might be ideal. If there is no explicit model, or a single static model and a distance metric is readily available, it might be preferred. Richter [18] defines a "neighborhood structure" as an integral part of a fitness landscape, so we believe evolutionary computation is a good environment for distance-based metrics.

Whatever the kind of metric used, is important to note that a higher value of novelty is not necessarily desirable. As a simple example, consider a set of observations consisting entirely of random noise, such as a "poem-generator" that simply generates long strings of random characters. These would have high novelty (either in the distance or learning sense), but it could hardly be called a poem generator. It is clear that both low-novelty and extreme-novelty can be undesirable to a system, which is why some authors define the *interestingness* of an object as a function relating its novelty to some desirability metric. A Wundt curve [19] is a hedonistic function commonly used to express this relationship [13] [12] [20]. In this sense, interestingness might be characterized by just the right amount of novelty - not too much, not too little.

Learning-based interestingness is also defined in a way to avoid excessively high novelty (unpredictability). Schmidhuber, as part of his theory of artificial curiosity, provides a comprehensive framework [21] for characterizing the learning progress of an agent by noting the intimate relationship between prediction and compression. An observation is termed interesting if it enables the agent to learn some previously unknown irregularity, that is, further compress the available data. Rehuis *et al.* [12] discuss a number of different learning-based interestingness metrics, which attempt to maximize the learning progress induced by including a new observation in the model: Actual Learning Progress, Previous Learning Progress, Previous Competence Change and Reducible Error. These are all based on the difference between the prediction error in a region of the problem space at two points in time.

The use of these terms (novelty, interestingness, value) is not entirely consistent across all literature. For this reason, we find it convenient to settle on some definitions for our

purposes, which lean closer to the way the terms are used in [12]. These definitions are:

Novelty: any measure of dissimilarity between a sample concept and a collection of concepts (distance-based novelty) or an expression of the prediction error of a surrogate model (learning-based novelty).

Interestingness: a function of how desirable a solution is based on its novelty. This will typically assign a low score both to low-novelty and excessive-novelty solutions.

Surprise: synonymous with learning-based novelty, that is, a measure of how much a candidate solution deviates from a model's expectation.

C. Value

Wiggins defines Value as "The property of an artifact (abstract or concrete) output by a creative system which renders it desirable in the context in which it is produced". Given that we also defined interestingness with regards to desirability, a closer look at the relationship between interestingness and value is necessary.

We define value as any measure of desirability, possibly domain-specific, while interestingness will be used solely as a more domain-agnostic measure of desirability that depends only on the underlying novelty metric and possibly the agent's internal state, but not on any externally assigned goals. To make the distinction clear, we propose an example inspired the space probe described in [20]. Imagine a space probe designed for mining some kind of ore in a distant planet. It has a number of sensors to measure some features of the world and is able of movement in four different directions. Via reinforcement learning, it uses data from its sensors to build a model that predicts the concentration of ore in parts of the world.

Consider now two regions of the world $R1$ and $R2$. At some point in time the model predicts a high concentration of ore in $R1$ and low concentration in $R2$. After exploring both regions, $R1$ is found to have low concentration of ore, $R2$ is found to have a high concentration and the model is updated. From a pure learning perspective (that is, in terms of learning progress), both observations can be equally useful. From a value perspective, it is clear that $R2$ has more value. $R1$ is only useful to the extent that, by exploring similar regions, the probe might eventually learn a new pattern that enables it to avoid such low-value regions in the future.

As Graziano puts it, a reinforcement learning agent can be given an "internal or curiosity reward", which directs its learning, and an "external reward", defined to achieve some pre-defined goal. These must be balanced against each other, as, unless the agent is provided with an accurate model from the start, it first needs to learn where the high-value regions are by exploring unknown (possibly low-value) regions. This is known as the exploration and exploitation problem.

A more classic formulation of the exploration and exploitation problem is given by the Multi-Armed Bandit (MAB) problem, in which a gambler is faced with N slot machines (also known as "one-armed bandits") with unknown reward distributions and must decide which machine to play at each point in time. An in-depth study of the MAB problem is outside the scope of this article. For more information, see [22]. In a Reinforcement Learning context, we will take novelty or interestingness (depending on the formulation of the problem) to be related to an agent's internal reward, encouraging exploration, and value to be related to an agent's external objective, encouraging exploitation. For a pure learning agent, an external definition of value might not be necessary.

Another interesting application of the relationship between novelty and value is seen in Lehman and Stanley's novelty-based evolution [15]. They implement novelty search as an extension of the NEAT method [23], using sparseness as metric for novelty, where distance is a domain-dependent measure of behavioral difference. Sparseness is, in turn, measured against the current population plus an archived set of high-novelty solutions. The novelty of a solution is used as selection factor for the evolving population, and the external objective is only used as a stopping condition test. By not using a fitness function based on the external objective, they outperform traditional methods in some deceptive environments, that is, where the fitness function leads too often to local optima. This indicates that when a good heuristic for the desired objective is unavailable, search through novelty alone can still lead to good results. Another possibility is a combined approach, where both novelty and traditional fitness are rewarded concurrently in a multi-objective formulation of the problem [24].

III. Games as Mixed-Initiative Research Platforms

The recent years have seen advancements both in systems that facilitate human creation and systems able of autonomous creation. However, researchers have noted a gap in systems that can work in tandem with one or more human agents, and achieve similar levels of initiative and responsibility as would be expected from a human partner. These are known as mixed-initiative systems. Some authors also use the term human-computer co-creativity, or mixed-initiative co-creativity, when emphasizing the creative nature of the output of such systems.

Carbonel [25] defines mixed-initiative systems as "one in which both humans and machines can make contributions to a problem solution, often without being asked explicitly". This notion is developed by Burstein and Mcdermott [9], who investigate how humans and machines can "best share information about and control of plan development" in a mixed-initiative system so that each agent works in areas where they perform best, use the appropriate representation for the communication of plans and have the means of acquiring and transferring authority over tasks. They identify six areas of AI research that needed to be addressed to enable their proposed model of mixed-initiative planning systems: plan-space search control management, representations and sharing of plans, plan revision management, planning and reasoning under uncertainty, learning and inter-agent communications and coordination.

Yannakakis *et al*. [7] identify "mixed-initiative co-creation (MI-CC) as the task of creating artifacts via the interaction of a

human initiative and a computational initiative", emphasizing the proactivity of the contributors, and differentiating it from "non-proactive computer support tools (e.g. spell-checkers or image editors)". They also argue that, if such a system is able to foster human creativity, then it can be called mixed-initiative co-creativity.

Krüger *et al.* [8] classify interaction between human and machine system in three levels of cooperation complexity: tools, adaptive tools and cooperative assistants. With a "tool" the human user has complete responsibility for the success of the operation and adaptation to different tasks. An "adaptive tool" has a model the environment to adapt to different situations, but has no capability to resolve possible mismatch between its goals and the humans goals. "Cooperative assistants" have a model not only of the environment, but of the human user, and are equipped with a transparent interface enabling the negotiation of responsibilities and goals. Although they do not use the term mixed-initiative, it is our view that such a cooperative assistant would qualify as mixed-initiative.

A similar distinction is drawn by Davis *et al.* [26], between what they call Creativity Support Tools (which support a creative person), Computational Creativity systems (which autonomously create products) and Computer Colleagues, which are "Co-creative agents (that) collaborate with humans in real-time improvisation to enrich the creative process". Davis [27] previously defined human-computer co-creativity as "a situation in which the human and computer improvise in real time to generate a creative product", where "the contributions of human and computer are mutually influential" and that "introduces a computer into this collaborative environment as an equal in the creative process". (Though one can of course think of useful co-creative processes where the computer is not an equal.)

Games have been considered the "killer app" for computational creativity [28], due to being multifaceted , content intensive, benefiting greatly from procedual generation techniques and rich (highly interactive and engaging). Games have also traditionally been used as benchmarks for AI. Of particular interest are general game-playing algorithms, which can in principle be applied to any games and better generalize to other real-world problems. For example, the GVGAI competition offers a set of 2D arcade-like games [29]. The use of games as AI benchmarks has received recent media attention due to the success of DeepMind's success at the game of Go with AlphaGo [30], AlphaGo Zero [31] by combining reinforcement learning and Monte Carlo tree search. This paradigm has also yielded success in other games by Anthony *et al.* [32] and by DeepMind's AlphaZero [33]. Games are also fun. Perez *et al.* [29] suggest that this leads to higher interest in AI research by the general public, and a 2014 review of gamification studies by Hamari *et al.* [34] concludes that, although some methodological issues were found, most studies yielded positive effects of gamification. We would like to investigate whether the use of game-like techniques can lead to the design of better co-creativity tools for real world problems.

Finally, we have identified several modern games where we believe a good AI controller, especially one designed for co-operative play with humans, would benefit from addressing many specific issues listed by mixed-initiative and co-creativity researchers as research topics for the development of the field. Tables I and II illustrate a mapping between these research topics and games that would serve as interesting problems for those research topics. We further detail the correspondence between research topics and games below:

Agent modeling: A lot of research in mixed-initiative systems and co-creativity is concerned with building a good model of the other agent's behavior and goals. For Burstein and Mcdermott [9], intent recognition (e.g. filling in the gaps of a plan that is not specified to the degree of atomic actions) and learning user preferences are important tasks of mixed-initiative planing systems. The ability to build a model of the user is one of the factors that distinguish a cooperative assistant from an adaptive tool for Krüger *et al.* [8].

Hadfield-Menel *et al.* [35] introduce Cooperative Inverse Reinforcement Learning (CIRL), a framework of cooperation between a Human H and a robot R, where both players are rewarded by the same reward function, which is known only by H. R tries to infer the reward function from H's actions. They show that when H tries to greedily maximize its own rewards, R might learn a poor approximation to the real reward function and achieve suboptimal results, so optimal solutions may involve active instruction by the human. The use of Generative Adversarial Networks (GANs) [36] to generate novel artifacts based on the design objectives of a user [37] or emulating a specific art style [38] is also a recent and promising approach to this problem.

The amount of time or data available for learning can also impose constraints on the techniques used. If a behavior must be learned over the course of a single game session, for example (rather than over a large number of games), one approach used by Barret *et al.* [39] is to pre-compute a set of strategies S and assume the other player is using strategy S_i with probability p_i, using Bayesian reasoning to update the probability of each strategy whenever the other player makes an action. The value of a prospective action with each possible paired strategy is weighted by their probability to determine the best action. They show this can lead to better results than simply mirroring the other player, even when the actual strategy is not one of the strategies contained in S.

Another useful technique is empowerment maximization [40], [41]. Empowerment is an information-theoretic, intrinsic motivation metric that formalizes how much potential causal influence an agent has upon the world it can perceive. An artificial agent motivated to maximize its human partner's empowerment could sidestep the issue of creating a complex model of the other agent's intentions by simply acting to leave their partner's options open.

In games, the need to predict the other player's actions and objectives arises naturally in competitive environments, especially those involving simultaneous action selection (like *Race for the Galaxy* (Tomas Lehmann, 2007) and other forms

TABLE I
RESEARCH TOPICS AND GAMES

Agent Modelling	Changing Environment	Nontrivial Goals		
		Emerging goals	Hidden Goals	Dynamic Goals
Poker Race for the Galaxy *Hanabi* *Magic Maze*	*Pandemic* *Flashpoint* *Overcooked*	Minecraft Roleplaying Games	Mafia Werewolf The Resistance *Shadows over Camelot** *Dead of Winter**	Ticket to Ride Terra Mystica *Pandemic Legacy: Season 1*

Mapping of research topic to games. Games in italics are cooperative. Games with an asterisk are cooperative with an optional traitor mechanic. Underlined games are electronic games

TABLE II
RESEARCH TOPICS AND GAMES (CONT.)

Asymmetric responsibilities	Communication	
	Unconstrained	Constrained
Pandemic *Magic Maze* *Can't Drive This*	*Pandemic* *Flashpoint*	*Hanabi* *Magic Maze* Real-Time Games (in general) Competitive Games (in general)

of bluffing (like Poker). In cooperative games, the need for agent modeling is alleviated if players are allowed to freely coordinate their actions. However, some cooperative games like *Hanabi* (Antoine Bauza, 2010) and *Magic Maze* (Kasper Lapp, 2017) enforce communication restrictions, which makes agent agent modeling a key factor for success.

Changing environment: Referring to traditional AI planning systems, Burstein and Mcdermott [9] state "the worlds in which these planners worked tended not to change much, fight back at all", and regards plan revisions and reasoning under uncertainty as two major areas of necessary research. For Krüger *et al.* [8], the ability to"change one or more of its own parameters in response to environmental variations" separates regular tools from adaptive tools, and is one of the requirements for cooperative assistants.

Many modern tabletop games excel in thematically representing environment changes inspired in real world uncertainties. In *Pandemic* (Matt Leacock, 2008), in during the Infection phase, cards are drawn from and infection deck to randomly add disease cube to cities in the board. If not treated timely by the players, these might induce chain reactions and defeat the players. *Flashpoint: Fire Rescue* (Kevin Lanzing, 2011) has the Advance Fire phase, where smoke and fire can be added to the board, which can cause explosions, structural damage to a collapsing building and knock down player-controlled Firefighter units. These phases usually occur in between player action phases to randomly provide either resources or obstacles to the players, and we term them environment phases for generality. In *Overcooked*, an electronic cooperative cooking game, the ingredients each player has access to changes with shifts in the map layout.

In some games, the goal of the game itself (that is, the scoring function) may change unpredictably during the course of the game (for example, limited-time scoring opportunities). We investigate these and other goal-related features below.

Nontrivial goals: Real-life goals are often nontrivial. They might be unknown to some of the agents, as in [35]. The goal might change during the execution of a project or parts of it may be implicitly specified [9]. The goal might be complex and broken into subtasks, and the responsibility for each subtask must be properly assigned, which could involve negotiation [8], [9]. In short, Davis *et al.* [26] characterize goals as "socially negotiated, dynamic and emergent".

In some "games"[1], such as *Minecraft* (Mojang, 2008) and roleplaying games, there is no overall objective stated by the rules, although the players might still define objectives for themselves based on what is fun for them, negotiate it with other players and attempt to achieve them via cooperation or competition. We term these games with regards to their goal as Emerging, due to their emergent nature as a product of the interaction between players and the environment.

Modern tabletop also employ many variations of secret objectives. Although we could not find a fully cooperative game with hidden goals, social deduction games such as *Werewolf* (Davidoff, Plotkin, 1986) and *The Resistance* (Don Eskridge, 2009) feature competition between two or more factions (whose members cooperate among themselves), where each player typically only know the allegiance of a small fraction of the other players (and thus, their objectives). *Shadows Over Camelot* (Cathala, Laget, 2005) and *Dead of Winter: A Crossroads Game* (Gilmour, Vega, 2014) are semi-cooperative games with a random probability of one player being assigned a traitor role. The mere possibility of a traitor encourages players to second-guess other player's reasons. Dead of Winter features a fairly unique mechanic where, even if no traitor is present, each player's goal is composed of a public objective, shared by all non-traitor players, and a secret objective, where a player only wins if the group fulfills the public objective and they personally fulfill their secret objective (so that one or more players might still lose even

[1]At this point, we want to acknowledge the controversy in calling these activities games. In *Rules of Play* [42], Salen and Zimmerman's definition of game involves there being a quantifiable outcome. We sidestep this discussion and call them games for simplicity and consistency with common usage.

if the group achieves success). This adds another layer of complexity where seemingly strange behavior by a player can be justified either by their secret objective or by a traitor role, and a non-traitor player's need to fulfill their secret objective might lead to the failure of the entire group.

Dynamic goals (where the scoring function itself changes over the course of a game) are also common: in *Ticket to Ride* (Alan Moon, 2004), players have the option of drawing extra objective cards, achieving extra score if they manage to fulfill these new objectives, at the risk of score penalties if they fail. In *Terra Mystica* (Drögemüller, Ostertag, 2012), a unique scoring tile is randomly drawn for each turn, enabling limited-time scoring opportunities for all players. A cooperative example is *Pandemic Legacy: Season 1* (Daviau, Leacock, 2015), a variation of *Pandemic* where players play missions in a persistent and evolving world, and a mission's objective may be altered mid-course by specific storyline events.

Asymmetric responsibilities and areas of expertise: For Burstein and Mcdermott [9], two of the high-level goal are to enable proper communication between agents with different areas of expertise, and that each agent works in areas where they perform best. Krüger *et al.* [8] gives an example of a cleaning robot that is able to identify areas where it cannot access (e.g. due to being blocked by an object) and proactively request assistant from the human user (who has a different set of skills and is able to e.g. move the object away). Different responsibilities (such as teaching and learning) can also be a result of asymmetric information, such as in [35].

In *Pandemic* and many other games, each player controls a unique character with special abilities, such as performing one specific type of action more efficiently. Some games are more radical in the variability between player powers. In *Magic Maze*, players share control of a group of character pawns, but each player can only move a pawn in one specific cardinal direction. In *Can't Drive This* (Pixel Maniacs, 2016), one player takes the role of a driver while the other dynamically builds the road on which the first player must drive.

Communication: Researchers highlight the need for a shared representation [9] or interface [8] in which communication can happen. Burstein and Mcdermott [9] also implicitly acknowledge a cost to associated with communication when stating "it is almost necessarily the case that details will be left out, if the communication is to be succinct enough to make it worth defining the task for another to carry out". Lu *et al.* [43] use a cooperative co-evolutionary approach to demonstrate how the frequency at which communication occurs impacts cooperative performance under different communication costs. Finally, the problem definition itself might disallow certain forms of communication, or allow no communication at all, in which case agents still can gain information by reasoning about other agents' actions [35].

Games offer an avenue for exploring all of these problems. In games that allow unrestricted communication, such as *Pandemic* and *Flash Point*, complex communication involving conditional logic and algorithm building can emerge, as shown by Berland [44]. Designing a communication scheme with comparable expressive power for effective human-AI and AI-AI cooperation is an open problem. In the Tiny Coop environment [45] communication actions are available, allowing each agent to signal the direction it would like its partner to move in the immediate future. However, human communication often happens not at the level of individual actions, but in terms of higher level goals and their dependencies. The need for communication can also be triggered by specific events, such as the completion of a goal or a change in environment. A recent example of development in this direction are by Schrodt *et al.* [46], whose agent is able to establish cooperative goals in a variant of *Super Mario Bros.* while thinking out loud its current intentions and state.

In other games, the communication is restricted by the game rules. In *Hanabi*, players can only communicate by expending a limited number of hints, which can only state the color or value of cards in another player's hand. In *Magic Maze*, players can freely communicate, but only at specific points in time. As a real-time game, time spent elaborating the plan comes at the cost of time for execution of the plan.

In some competitive games, the rules allow full communication, including negotiation and partnerships, but it must occur in full view of other players. In this scenario, the cost of communication is the information that is leaked to antagonist players, and so communication is a strategic decision.

IV. Metrics for Co-creative Agents

We propose the following types of metrics for co-creative agents in game environments:

- **Value:** For any game with fixed objectives stated by the rules, a natural way to measure value is the game's scoring function. For games with emerging or hidden goals, explicit feedback from the user, if available, can also be used as a value metric. For procedurally generated content, value could be measured by a pre-determined fitness function of the generated artifact's features (as seen in [7]), by results of simulation [47] or by subjective evaluation of the human player, who selects their preferred generated artifact [7].
- **Learning-based metrics:** An agent might attempt to build a model of the other agents over the course of a game session or multiple sessions. A model of the user's behavior can be used to predict their action in a tree search algorithm. A model of the user's preferences can be used to predict the probability of acceptance of an artifact by the user. The accuracy of these predictions is a metric of learning-based novelty, and the higher the confidence of the model in a result, the higher the surprise if the prediction fails.

  For an agent attempting to gradually build a model of a player, care must be taken to isolate gains in performance (either in terms of value or in terms of accuracy of predictions) due to improvements on the part of the agent and improvements on the part of the human. After all, a human player could play with a simple, non-learning agent, and the agent could still report an increase in

performance due to the human player better learning to play the game or play matching the agent's expectations. A statistical analysis of player improvement over time is given in [47].

To avoid this confounding factor, it is important implement baseline agents, with statistically unchanging behavior, who would serve as proper control groups when paired with learning agents and humans, so that the impact of an agent's learning on performance cannot be overestimated.

- **Distance-based metrics:** In some scenarios, the product of each decision by the agent will not be a single, atomic action, but a number of options for the other agents to choose from, such as a number of action plans or a number of in-game artifacts for use of the human player. In these cases, distance-based metrics of novelty and interestingness can be used to make sure the suggestions offer a varied sample of the decision space, rather than small variations of a single idea. That way, the user is most likely to find a suggestion they identify it, and tweak some finer details to their own preference.
- **Empowerment metrics:** Empowerment "grows when different actions lead to different perceivable outcomes", and is a form of intrinsic motivation [41]. As such, it can be used in the absence of explicitly stated goals. We believe empowerment can also be used to maximize chance of acceptance of a suggestion, similar to distance-based metrics, by providing many relevant choices to the user.
- **Communication metrics:** The most direct way to measure the effectiveness of a communication scheme is simply to measure the difference in value (or in accuracy of predictions) achieved by cooperating under different schemes (or with no communication), as is done in [45]. The frequency of communication [43] can also be an important metric in scenarios where there is a communication cost or where player experience could be negatively impacted by a high-frequency stream of low-level communication actions.

The proposed metrics pose an initial approach to quantify the success of co-creative agents in cooperative games and similar environments.

V. The way forward

In tables I and II, we listed some characteristics of games that provide interesting research topics for human-computer cooperation. In our view, the most promising application scenarios are those focused on **agent modeling** and **communications** and are the biggest gap in current cooperative systems. They are at the core of co-creative cooperative activities, while the remaining entries of the table serve as challenges to be addressed by better cooperative systems (which include agent modeling and communication as core components): how will the other player react by a change in the environment? How to infer an unknown goal from a player's actions? How to communicate a change of plans due to a change in the environments? How to communicate (or predict) which activities are to be performed by each agent, especially under time constraints?

Going forward, we believe communication and agent modeling can also feed off each other. On one hand, an agent can use its ability to communicate to build more accurate models, either by directly asking for missing information or by picking up on cues from information provided by its peers. On the other hand, having an accurate model can help determine what information to share or ask for. A very clear example of this dynamic is in the game *Hanabi*, where different players are comfortable operating under different levels of implicit information (e.g. how willing are they to risk playing a card with incomplete information?). Observing the hints given by a player can help us infer how much information they need for their own actions, while knowing how they act under uncertainty helps us determine what hints to give.

While section III provides many examples of application scenarios to achieve progress in human-machine cooperation in the short term, our long term view is that this research can lead to applications where high-level goals and plans can be negotiated between human and artificial agents, taking into account their specific abilities and knowledge. The artificial agents will then be able to fill in small gaps in the plan by reasoning about a model of the world and of the other agents. Alternatively, the agent can proactively request any information it is missing if the gaps are too large to be filled.

It will be able to detect events that require a change of plans (such as a change in environment, available resources or goals of the group) and once again communicate and negotiate the new plan. All along the process, novel and valuable artifacts will be produced through computational creativity techniques, where novelty and value are judged in regards to a model of the knowledge and preferences of the target audience.

CONCLUSION

We started this paper by asking what game-like environments would be ideal for measuring the impact and success of co-creative cooperative agents. We answer that question by proposing several types of metrics, based on a thorough research on computational creativity and metrics used in the computational intelligence community for the related concepts of novelty, value, interestingness and surprise. We have shown how research in these scenarios, and similar games, can help shed light on open questions of the field and provided a vision of how these systems could operate in the long term.

We hope that this can lead to the development of better mixed-initiative, co-creative systems for a variety of domains, including industrial applications, where human and machine can cooperate working in areas where they perform best, communicating efficiently to achieve nontrivial goals under a changing, uncertain environment.

ACKNOWLEDGMENT

Rodrigo Canaan gratefully acknowledges the financial support from Honda Research Institute Europe (HRI-EU). We

would also thank Nikolas Dahn (TU Ilmenau) and Dr. Thomas Weisswange (HRI-EU) for their valuable input to this research.

REFERENCES

[1] M. A. Boden, "Creativity and artificial intelligence," *Artificial Intelligence*, vol. 103, no. 1-2, pp. 347–356, 1998.

[2] S. Colton, G. A. Wiggins *et al.*, "Computational creativity: The final frontier?" in *ECAI*, vol. 12, 2012, pp. 21–26.

[3] P. McCorduck, *Aaron's code: meta-art, artificial intelligence, and the work of Harold Cohen*. Macmillan, 1991.

[4] S. Colton, "Creativity versus the perception of creativity in computational systems." in *AAAI spring symposium: creative intelligent systems*, vol. 8, 2008.

[5] ——, "The painting fool: Stories from building an automated painter," in *Computers and creativity*. Springer, 2012, pp. 3–38.

[6] C. Guckelsberger, C. Salge, and S. Colton, "Addressing the why? in computational creativity: A non-anthropocentric, minimal model of intentional creative agency," in *Proceedings of the Eight International Conference on Computational Creativity*, 2017.

[7] G. N. Yannakakis, A. Liapis, and C. Alexopoulos, "Mixed-initiative co-creativity." in *FDG*, 2014.

[8] M. Krüger, C. B. Wiebel, and H. Wersing, "From tools towards cooperative assistants," in *Proceedings of the 5th International Conference on Human Agent Interaction*. ACM, 2017, pp. 287–294.

[9] M. H. Burstein and D. V. McDermott, "Issues in the development of human-computer mixed-initiative planning," in *Advances in Psychology*. Elsevier, 1996, vol. 113, pp. 285–303.

[10] G. A. Wiggins, "A preliminary framework for description, analysis and comparison of creative systems," *Knowledge-Based Systems*, vol. 19, no. 7, pp. 449–458, 2006.

[11] G. Ritchie, "Some empirical criteria for attributing creativity to a computer program," *Minds and Machines*, vol. 17, no. 1, pp. 67–99, 2007.

[12] E. Reehuis, M. Olhofer, M. Emmerich, B. Sendhoff, and T. Bäck, "Novelty and interestingness measures for design-space exploration," in *Proceedings of the 15th annual conference on Genetic and evolutionary computation*. ACM, 2013, pp. 1541–1548.

[13] R. Saunders, P. Gemeinboeck, A. Lombard, D. Bourke, and A. B. Kocaballi, "Curious whispers: An embodied artificial creative system." in *ICCC*, 2010, pp. 100–109.

[14] T. Hester and P. Stone, "Intrinsically motivated model learning for a developing curious agent," in *Development and Learning and Epigenetic Robotics (ICDL), 2012 IEEE International Conference on*. IEEE, 2012, pp. 1–6.

[15] J. Lehman and K. O. Stanley, "Abandoning objectives: Evolution through the search for novelty alone," *Evolutionary computation*, vol. 19, no. 2, pp. 189–223, 2011.

[16] L. Itti and P. Baldi, "Bayesian surprise attracts human attention," *Vision research*, vol. 49, no. 10, pp. 1295–1306, 2009.

[17] S. Kullback, *Information theory and statistics*. Courier Corporation, 1997.

[18] H. Richter, "Analyzing coevolutionary games with dynamic fitness landscapes," in *Evolutionary Computation (CEC), 2016 IEEE Congress on*. IEEE, 2016, pp. 609–616.

[19] W. M. Wundt, *Grundzüge de physiologischen Psychologie*. W. Engelman, 1874, vol. 1.

[20] V. Graziano, T. Glasmachers, T. Schaul, L. Pape, G. Cuccu, J. Leitner, and J. Schmidhuber, "Artificial curiosity for autonomous space exploration," *Acta Futura*, vol. 4, pp. 41–51, 2011.

[21] J. Schmidhuber, "Formal theory of creativity, fun, and intrinsic motivation (1990–2010)," *IEEE Transactions on Autonomous Mental Development*, vol. 2, no. 3, pp. 230–247, 2010.

[22] S. Agrawal and N. Goyal, "Analysis of thompson sampling for the multi-armed bandit problem," in *Conference on Learning Theory*, 2012, pp. 39–1.

[23] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary computation*, vol. 10, no. 2, pp. 99–127, 2002.

[24] J.-B. Mouret, "Novelty-based multiobjectivization," in *New horizons in evolutionary robotics*. Springer, 2011, pp. 139–154.

[25] J. R. Carbonell, "Mixed-initiative man-computer instructional dialogues." BOLT BERANEK AND NEWMAN INC CAMBRIDGE MASS, Tech. Rep., 1970.

[26] N. M. Davis, Y. Popova, I. Sysoev, C.-P. Hsiao, D. Zhang, and B. Magerko, "Building artistic computer colleagues with an enactive model of creativity." in *ICCC*, 2014, pp. 38–45.

[27] N. Davis, "Human-computer co-creativity: Blending human and computational creativity," in *Ninth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2013.

[28] A. Liapis, G. N. Yannakakis, and J. Togelius, "Computational game creativity." in *ICCC*. Citeseer, 2014, pp. 46–53.

[29] D. Perez-Liebana, S. Samothrakis, J. Togelius, T. Schaul, S. M. Lucas, A. Couëtoux, J. Lee, C.-U. Lim, and T. Thompson, "The 2014 general video game playing competition," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 8, no. 3, pp. 229–243, 2016.

[30] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[31] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, p. 354, 2017.

[32] T. Anthony, Z. Tian, and D. Barber, "Thinking fast and slow with deep learning and tree search," in *Advances in Neural Information Processing Systems*, 2017, pp. 5366–5376.

[33] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel *et al.*, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," *arXiv preprint arXiv:1712.01815*, 2017.

[34] J. Hamari, J. Koivisto, and H. Sarsa, "Does gamification work?–a literature review of empirical studies on gamification," in *System Sciences (HICSS), 2014 47th Hawaii International Conference on*. IEEE, 2014, pp. 3025–3034.

[35] D. Hadfield-Menell, S. J. Russell, P. Abbeel, and A. Dragan, "Cooperative inverse reinforcement learning," in *Advances in neural information processing systems*, 2016, pp. 3909–3917.

[36] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[37] P. Bontrager, W. Lin, J. Togelius, and S. Risi, "Deep interactive evolution," *arXiv preprint arXiv:1801.08230*, 2018.

[38] A. Elgammal, B. Liu, M. Elhoseiny, and M. Mazzone, "Can: Creative adversarial networks generating art by learning about styles and deviating from style norms, june 2017," *arXiv preprint arXiv:1706.07068*.

[39] S. Barrett, P. Stone, and S. Kraus, "Empirical evaluation of ad hoc teamwork in the pursuit domain," in *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*. International Foundation for Autonomous Agents and Multiagent Systems, 2011, pp. 567–574.

[40] C. Salge, C. Glackin, and D. Polani, "Empowerment–an introduction," in *Guided Self-Organization: Inception*. Springer, 2014, pp. 67–114.

[41] C. Guckelsberger, C. Salge, R. Saunders, and S. Colton, "Supportive and antagonistic behaviour in distributed computational creativity via coupled empowerment maximisation," in *Proceedings of the Seventh International Conference on Computational Creativity*, 2016.

[42] K. Salen and E. Zimmerman, *Rules of play: Game design fundamentals*. MIT press, 2004.

[43] X. Lu, S. Menzel, K. Tang, and X. Yao, "Cooperative co-evolution based design optimisation: A concurrent engineering perspective," *IEEE Transactions on Evolutionary Computation*, 2017.

[44] M. Berland and V. R. Lee, "Collaborative strategic board games as a site for distributed computational thinking," *Developments in current game-based learning design and deployment*, vol. 285, 2012.

[45] J. Walton-Rivers, "Controlling co-incidental non-player characters," http://www.iggi.org.uk/assets/IGGI-2016-Piers.pdf access on 02/01/2018.

[46] F. Schrodt, Y. Röhm, and M. V. Butz, "An event-schematic, cooperative, cognitive architecture plays super mario," *Cognitive Robot Architectures*, vol. 10, pp. 10–15, 2017.

[47] A. Isaksen and A. Nealen, "A statistical analysis of player improvement and single-player high scores." in *DiGRA/FDG*, 2016.

Modern Techniques for Ancient Games

Cameron Browne
Games and AI Group
Department of Data Science and Knowledge Engineering (DKE)
Maastricht University
Maastricht, The Netherlands
cameron.browne@maastrichtuniversity.nl

***Abstract*—Games potentially provide a wealth of knowledge about our shared cultural past and the development of human civilisation, but our understanding of early games is incomplete and often based on unreliable reconstructions. This paper describes the Digital Ludeme Project, a five-year research project currently underway that aims to address such issues using modern computational techniques.**

***Index Terms*—ancient games, ludemes, general game playing, phylogenetics, history of mathematics, digital archaeoludology**

I. Introduction

The development of games has gone hand-in-hand with the development of human civilisation [1]. However, our knowledge of early games is incomplete and based on often unreliable interpretations of available evidence.

There has been a wealth of traditional game studies over recent centuries, from historical, anthropological, archæological – and more recently ethnological and cultural – perspectives. There is now also a wealth of computational game studies; games have always been a driving factor behind *artificial intelligence* (AI) and *machine learning* (ML) research since the inception of these fields in the 1950s [2], and now Game AI is maturing as a research field in its own right [3].

However, there has been little overlap between computational and historical studies of traditional games. This paper outlines a newly launched research project aimed to address this gap, so that our historical understanding of games might benefit from current advances in technology.

A. The Digital Ludeme Project

The Digital Ludeme Project[1] is a five-year research project being conducted at Maastricht University over 2018–23, funded by a European Research Council (ERC) Consolidator Grant. The objectives of the project are to:

1) *Model* the full range of traditional strategy games in a single, playable digital database.
2) *Reconstruct* missing knowledge about traditional strategy games with an unprecedented degree of accuracy.
3) *Map* the development of traditional strategy games and explore their role in the development of human culture and the spread of mathematical ideas.

An ultimate goal of the project is to produce a "family tree" of the world's traditional strategy games, with which the dispersal of games and related mathematical ideas might be traced throughout recorded history. This will pioneer a new field of study called *digital archæoludology* (DA) which will involve the use of modern computational techniques for the analysis and reconstruction of traditional games from incomplete descriptions, to provide new tools and techniques for improving our understanding of their development.

This paper describes the project's research context, scope of games to be covered, the methodology used, and some potential benefits.

Funded by a €2m European Research Council (ERC) Consolidator Grant.
[1]http://www.ludeme.eu

II. Research Context

While there is much archæological evidence of ancient games, the rules for playing them are usually lost [8] and must be reconstructed by modern historians according to their knowledge of the cultures in which they were played [9], [10]. The rules for ancient and early games were typically passed on through oral tradition rather than being transcribed, which may have contributed to their variation and embellishment into the range of games that we see today [8], but means that our understanding of early games is largely based on modern reconstructions. The following examples demonstrate some of the issues involved.

Fig. 1. Ancient Egyptian hieroglyphic art showing Senet being played.

A. Missing Knowledge

Many boards and sets of playing pieces have been found for the ancient Egyptian game of Senet dating back to c.3500BC, some in pristine condition, allowing historians to deduce with reasonable certainty what type of game it was [11]. However,

the only known clues as to *how* Senet was played are found in hieroglyphic art dating back to c.3100BC, such as Fig. 1), which shows stylised characters playing the game.

Game historian H.J.R. Murray declined to propose a complete set of rules for Senet in his classic 1952 book [8]. Kendall did so in 1978 [12], but his version is based on snippets of information from sources spanning thousands of years and remains questionable. For example, Murray interpreted board squares marked with certain sacred Egyptian symbols as entry points for pieces, while Kendall interpreted these as points of departure from the game [13].

B. Loss of Knowledge

Even when records of the rules for ancient games *are* found, interpretation can be problematic. For example, the earliest known record of a game's rule set is for the Royal Game of Ur. Game sets were uncovered in Iraq dating back to 2600–2400BC, but it was not until Irving Finkel's 1990 study of two Sumerian stone tablets dating from 177–176BC [14] that the game's (probable) rules were found.

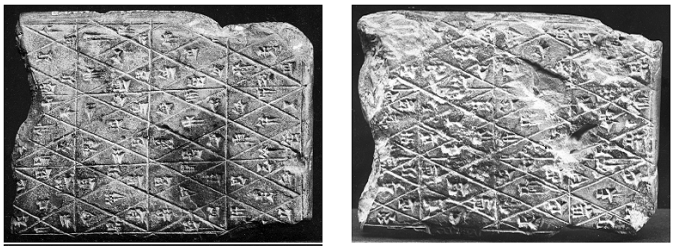


Fig. 2. Sumerian tablet showing game rules destroyed in World War II.

The first tablet was found by Finkel among 130,000 such tablets held in the British Museum. The second tablet (Fig. 2) was luckily photographed shortly before its destruction in a Parisian studio during World War II and recognised by Finkel half a century later [14]. Note that Finkel's interpretation of the game was made thousands of years after the tablets were made, which was itself thousands of years after the game was originally played.

In addition to wanton destruction caused by war, vandalism, looting, desecration, etc., more benign forces such as erosion and urban development can also take their toll on archæological evidence. For example, Crist describes the case of ancient game boards etched into rock surfaces in Azerbaijan [15], which were destroyed to make way for a housing development between one research trip and the next.

C. Translation Errors

Translation errors are another issue. Consider the game of Hnefatafl, played by Vikings from c.400BC and spread wherever they travelled, for which no known documentation of the original rules exists. A modern set is shown in Fig. 3

In 1732, Swedish naturalist Carolus Linnæus observed the related game Tablut being played in Lappland, and recorded its rules in Latin in his travel diary [16]. This account was translated by E. L. Smith in 1811, who mistook the phrase

Fig. 3. A modern version of the Viking game Hnefatafl.

"likewise the king" to mean "except the king", to produce a biased rule set that greatly favoured the king's side [17]. Murray used this translation as the basis for his 1913 reconstruction of Hnefatafl [18], making it the definitive rule set for many years [8], until players and researchers subsequently corrected this flaw to give the many versions of the game played today.

D. Reconstruction Errors

Often the archæological evidence of games provides little clue as to their rules, which must be deduced almost entirely from context. For example, the rock etching shown in Fig. 4 (left) was unearthed in Assos, Turkey, and estimated to be around 2,300 years of age [19]. This design (right) is listed as design #88 in Uberti's census of Merels boards [20], which revealed around 100 designs from more than 2,500 examples across 43 countries.

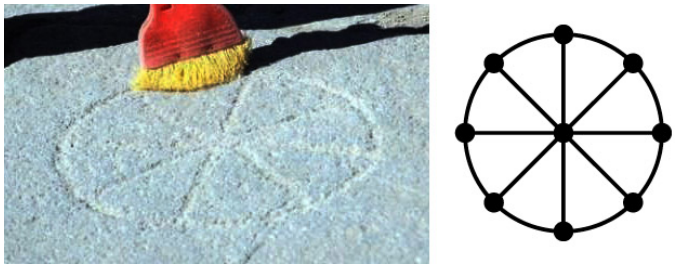


Fig. 4. Engraving found at Assos (Turkey) and a Small Merels board.

It is assumed that this board was used to play Round Merels,[2] which seems to be the default assumption for boards of this design, location and epoch. German historian Carl Blümlein proposed a plausible reconstruction of the rules in 1918, which became the accepted standard and was not questioned for almost 100 years, when a 2014 analysis revealed a critical flaw that allowed players to exploit infinite cycles [21]. It is now questioned whether this design was used for a different type of game or was not even a game at all.

E. Transcription Errors

Transcription errors can be an issue even with the records of more recent games. For example, the design shown in Fig. 4

[2]A miniature version of the traditional Merels or Nine Men's Morris.

(right) – a wheel with eight spokes – is also used for the 18th century Maori game Mu Torere from New Zealand, even though its rules are quite different to Round Merels.

Ethnomathematician Marcia Ascher noted in her 1987 study of Mu Torere [22] that at least two historians' transcriptions of its rules simplified out an apparently trivial starting rule, without which the game became unplayable. A full game tree expansion by mathematician Philip Straffin [23] (Fig. 5) shows that either player can win on the first move if this rule is not used, which is obvious from even the simplest analysis.

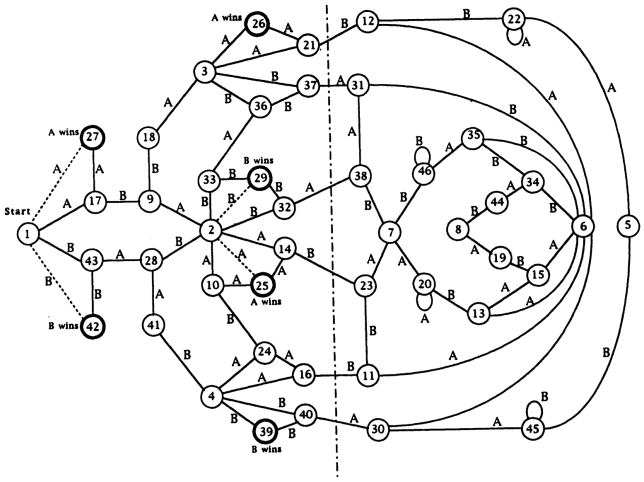


Fig. 5. Full game tree expansion of Mu Torere showing trivial wins (dotted).

F. Reinvention Estimates

The discovery of similar game boards in India and ancient Mexico – Pachisi and Patolli (Fig. 6) – was used in 1879 as evidence of early pre-Columbian contact between Asia and South America [24], even though the rules for each game are quite different [25]. This claim was disputed half a century later due to the notion of "limited possibilities" in design making coincidental reinvention more plausible [26], even though Murray [8] points out that such independent reinvention is generally unlikely.

Fig. 6. Similar boards for Pachisi (left) and Patolli (right).

Which view is more likely to be correct? Such analyses will remain speculation until methods are developed to provide quantitative evidence for such cases. A more accurate and complete knowledge of the development and spread of traditional games could help clarify such cases, and shed new light on trade routes and points of contact between cultures.

G. Partial Evidence

A challenging task facing historians is to reconstruct the rules of games when only some of the equipment is known. For example, Fig. 7 shows a game board and pieces dated to 375AD and found in 2006 in Poprad, Slovakia, in the tomb of a Germanic chieftain who served in the Roman army [27]. This equipment has no clear precedent in Europe, and historian Ulrich Schädler describes the reconstruction of the game's original rules as "impossible", as the board is incomplete and only a few playing pieces survived.[3]

Fig. 7. Partial game set found in a Slovakian tomb.

These examples – and many others from around the world – highlight the difficulty of compiling reliable knowledge of traditional games. What little evidence does exist is fragile and easily lost, and attempts to reconstruct missing rule sets have so far relied heavily on historical context rather than mathematical evaluation, but once accepted into the canon become the *de facto* standards. Thus, our knowledge of this important part of our cultural heritage is at best partial, or skewed by unreliable reconstructions.

While much attention has been paid to ensuring the historical *authenticity* of reconstructions, there has been to date no systematic approach to evaluating the *quality* of proposed reconstructions as games. This project aims to develop tools and methods for improving our understanding of traditional games with unprecedented mathematical rigour.

III. Scope

The Digital Ludeme Project deals with *traditional games of strategy*, i.e. games with no proprietary owner [4, p.5] that exist in the pubic domain,[4] and in which players succeed through mental rather than physical acumen. This category includes

[3]Conversation at the Board Game Studies Colloquium, Athens, April 2018.

[4]The more precise distinction between traditional games and those invented by known individuals and distributed by games companies [5] can lead to ambiguous cases [6].

most board games, some card games, some dice games, some tile games, etc., and may involve non-deterministic elements of chance or hidden information as long as strategic play is rewarded over random play. It excludes dexterity games, social games, sports, video games, etc.

This study will cover the full range of traditional strategy games throughout recorded human history, i.e. from around 3500BC, from all countries and cultures worldwide. Within this context, it is useful to distinguish the approximate time periods shown in Fig. 8:

- *Ancient*: before 500AD.
- *Early*: 500AD – 1500AD.
- *Modern*: after 1500AD.

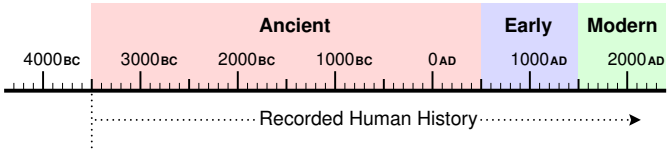


Fig. 8. Timeline of key periods in recorded human history.

In general, the older a game is, the less is known about it. The original rules are known for most modern games, some early games, but few ancient games.

A. Influencers

It would be unrealistic to attempt to model *every* traditional strategy game. For example, of the thousands of known Chess variants,[5] hundreds could fall under the umbrella of "traditional". There also exist over 800 known variants of Mancala, let alone the undocumented ones [7].

It is difficult to even estimate the number of known traditional strategy games. The BoardGameGeek (BGG) online database[6] lists around 100,000 known board games with (probable) invention dates and details regarding designer and publisher. However, entire families of games, such as the hundreds of Mancala variants, are typically collapsed into a few representative entries, making the total number of BGG entries a gross underestimate of the number of actual games, possibly by orders of magnitude.

Instead, the Digital Ludeme Project will investigate a representative sample of 1,000 of the world's traditional strategy games, which include the most influential examples throughout history. Such *influencers* might be identified by:

- *Appeal:* Estimated total number of players.
- *Impact:* Number of similar games that follow.
- *Importance:* Footprint in the literature.

The idea is to focus on those games that are most important to the evolution of traditional strategy games. Games that are known to have existed, but which might have only been played within one community or even one family, and for which there is no evidence of any influence over later games, constitute evolutionary dead ends that are of less interest for this task.

[5]http:// www.chessvariants.com
[6]http://www.boardgamegeek.com

IV. Ludemes

Games will be modelled as structures of *ludemes*, i.e. game memes or conceptual units of game-related information [28]. These constitute a game's underlying building blocks, and are the high-level conceptual terms that human designers use to understand and describe games. Previous work on evolving board games [29] demonstrated the effectiveness of the ludemic model for the automated generation of games.

Table I shows how the game of Tic-Tac-Toe might be described in ludemic form. This description is simple, clear, and encapsulates key concepts and labels them with meaningful names. Breaking games down into ludemes makes them easier to model, compare and manipulate digitally, and makes it possible the model the full range of traditional games in a single playable database.

TABLE I
Ludemic Description of Tic-Tac-Toe

```
(game Tic-Tac-Toe
    (players White Black)
    (equipment
        (board (square 3) diagonals)
    )
    (rules
        (play (add (piece Own) (board Empty)))
        (end (win All (line 3 Own Any)))
    )
)
```

A. Stanford GDL

Table II shows the same game described in the Stanford Logic Group's *game description language* (GDL), which has become the standard method for describing games in *general game playing* (GGP) research [30]. GDL offers benefits of transparency (the game description itself contains the instructions for updating the game state) and correctness checking (that the rules are well-formed).

By contrast, the ludemic approach hides the implementation details to provide simplicity, encapsulation, efficiency and ease of use. For example, if we wish to modify Tic-Tac-Toe so that players aim to make a line-of-4 on a 5×5 board, or play on a hexagonal grid, or aim for some other winning condition altogether, then each of these changes would involve a trivial parameter adjustment or swapping of keywords in ludemic form. Implementing these changes in GDL format, however, would require a significant amount of the code to be rewritten and retested. The encapsulation of concepts makes ludemic descriptions easier to modify and more evolvable than GDL descriptions.

V. Ludii System

A complete *general game system* (GGS) for modelling, playing, analysing, optimising and generating the full range of traditional strategy games is being developed for this project. This system, called Ludii, is based on similar principles to

TABLE II
GDL DESCRIPTION OF TIC-TAC-TOE

```
(role white)
(role black)
(init (cell 1 1 b))
(init (cell 1 2 b))
(init (cell 1 3 b))
(init (cell 2 1 b))
(init (cell 2 2 b))
(init (cell 2 3 b))
(init (cell 3 1 b))
(init (cell 3 2 b))
(init (cell 3 3 b))
(init (control white))
(<= (legal ?w (mark ?x ?y))(true (cell ?x ?y b)) (true (control ?w)))
(<= (legal white noop) (true (control black)))
(<= (legal black noop) (true (control white)))
(<= (next (cell ?m ?n x)) (does white (mark ?m ?n)) (true (cell ?m ?n b)))
(<= (next (cell ?m ?n o)) (does black (mark ?m ?n)) (true (cell ?m ?n b)))
(<= (next (cell ?m ?n ?w))(true (cell ?m ?n ?w)) (distinct ?w b))
(<= (next (cell ?m ?n b)) (does ?w (mark ?j ?k)) (true (cell ?m ?n b))
    (or (distinct ?m ?j) (distinct ?n ?k)))
(<= (next (control white))(true (control black)))
(<= (next (control black))(true (control white)))
(<= (row ?m ?x)(true (cell ?m 1 ?x))(true (cell ?m 2 ?x))(true (cell ?m 3 ?x)))
(<= (column ?n ?x)(true(cell 1 ?n ?x))(true(cell 2 ?n ?x))(true(cell 3 ?n ?x)))
(<= (diagonal ?x)(true (cell 1 1 ?x))(true (cell 2 2 ?x))(true (cell 3 3 ?x)))
(<= (diagonal ?x)(true (cell 1 3 ?x))(true (cell 2 2 ?x))(true (cell 3 1 ?x)))
(<= (line ?x) (row ?m ?x))
(<= (line ?x) (column ?m ?x))
(<= (line ?x) (diagonal ?x))
(<= open (true (cell ?m ?n b))) (<= (goal white 100) (line x))
(<= (goal white 50) (not open) (not (line x)) (not (line o)))
(<= (goal white 0) open (not (line x)))
(<= (goal black 100) (line o))
(<= (goal black 50) (not open) (not (line x)) (not (line o)))
(<= (goal black 0) open (not (line o)))
(<= terminal (line x))
(<= terminal (line o))
(<= terminal (not open))
```

the previous LUDI system [31], but improved in almost every way to be more general, extensible and efficient.

The core of LUDII is a *ludeme library* consisting of a number of Java classes each implementing a particular ludeme. Games are described as structured sets of ludemes, as per Table I, according to an EBNF-style grammar automatically generated from the ludeme library's class hierarchy using a *class grammar* approach [32]. Game descriptions can then be compiled directly to Java byte code according to their underlying ludeme classes.

A. Plausible AI

AI move planning will be performed using *Monte Carlo tree search* (MCTS) [33] with playouts biased by strategies learnt through self-play. MCTS has become the preferred approach for general game playing over recent years, due to its ability to devise plausible actions in the absence of any tactical or strategic knowledge about the given task. Although it can prove weaker for some games than others, it provides a good baseline level of AI play for most games.

The combination of deep learning with MCTS has recently had spectacular success with Go [34]. However, this level of superhuman performance is not required for this project, where a more modest level of play pitched just beyond average human level is preferable, in order to estimate the potential of games to interest human players. Superhuman AI that plays differently to humans could actually bias evaluations; instead, we want an AI that makes moves that human players would plausibly make.

1) Lightweight Local Features: To elevate MCTS to a sufficient level of play for all games, playouts will be biased with domain-dependent information in the form of lightweight features that capture geometric piece patterns, learnt through self-play. For example, the pattern shown in Fig. 9, which completes a threatened connection in connection games played on the hexagonal grid, improves MCTS playing strength when incorporated into the playouts of such games [35].

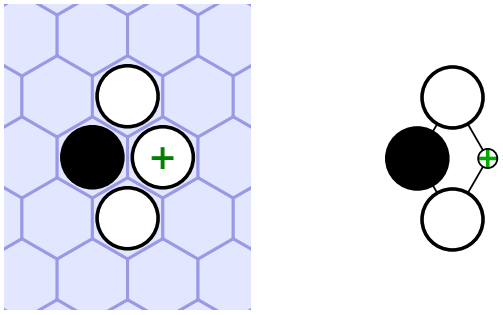


Fig. 9. A strong pattern for connection games on the hexagonal grid.

Such patterns represent local strategies that human players typically learn to apply. They will not capture more complex global strategies, but should serve to improve MCTS to plausible levels of play, and – importantly – could give an indication of a game's strategic potential.

B. Game Evaluation

When evaluating rule sets, it is important to consider the *quality* of the resulting games, which is the vital element missing from many historical studies of games. If a rule set is flawed, or does not have potential to interest human players, then it is unlikely that is how the game was actually played.

Previous work [31], [37] has outlined robust indicators of flaws in games that can be easily measured through self-play:

1) *Length:* Games should not be too short or long.
2) *Fairness:* Games should not unduly favour either player.
3) *Drawishness:* Games should not end in draws too often.

The question of what makes a game "good" in players' eyes is much more difficult; there are no universal indicators of game quality, and preferences can differ between individuals, cultures, and across time. However, it would make sense that a key quality for strategy games should be their *strategic depth*, indicated by the number and complexity of potential strategies that players can learn.

Lantz *et al.* propose the notion of the *strategy ladder* [38]. Fig. 10 shows three plots that represent three different games,

with dots indicating relevant strategies that players can learn. The leftmost game (white dots) is uninteresting as it has a simple winning strategy that is easy to learn. The rightmost game (white dots) is uninteresting as it has difficult strategies that are too hard to learn. The middle game (black dots) has a variety of strategies of linearly increasing complexity; the player can immediately see some simple strategies, but learn more complex strategies as the game is played more.

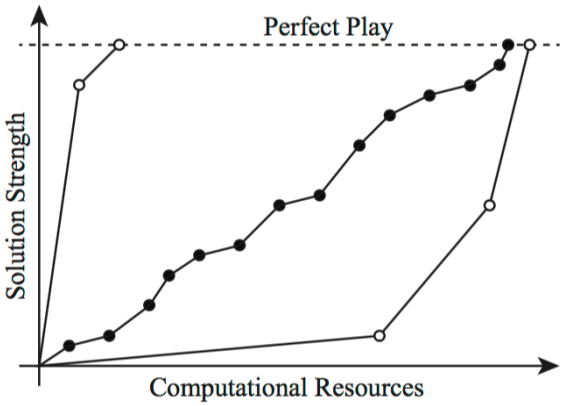


Fig. 10. Linear acquisition of strategies in an interesting game (black) [38].

Strategic depth should be considered relative to a game's complexity. For example, Mu Torere (46 legal positions [23]) could be expected to involve fewer strategies than Go ($\approx 2.08 \times 10^{170}$ legal positions [39]). Other ways to estimate the strategic depth of a game might include comparing relative win rates over a range of AI agents of varying strength.

C. Strategy Learning, Transfer and Explanation

If these lightweight features based on piece patterns represent local strategies, then the number and complexity of learnt features could give an indication of a game's strategic depth. Basing local piece patterns on the adjacency of a game's underlying graph (Fig. 9, right) rather than the board itself (Fig. 9, left) provides geometric independence that allows learnt features to be transferred between different board types. Keeping the feature attributes as simple and abstract as possible makes it more likely that features might also be transferrable to other game types.

The fact that ludemes are labelled with meaningful names raises the possibility of automatically explaining learnt strategies in human-comprehensible terms. For example, the strategy encoded in Fig. 9 might be explained as "complete threatened connections between your pieces". The strategy encoded in Fig. 11, effective for the recent game Quantum Leap [36], might be explained as "make thin groups of your pieces", by encouraging the growth of singletons and the extension of adjacent pieces except at mutually adjacent points.

VI. Genetics of Games

In order to map the dispersal of traditional strategy games, it is useful to cast the mechanism for their evolution into a biological genetic framework. Anthropologist Alex de Voogt has stated: *There is nothing genetic about board games. There are no genes or mental parameters that only change with a new generation of people as in linguistics or in biology* [7, p.105]. However, I would argue that the ludemic model allows us to distinguish between the *form* of a game defined by its ludemic makeup of rules and equipment (i.e. genotype) and the *function* of a game defined by the behaviour it exhibits when played (i.e. phenotype). Ludemes are the "DNA" that define each game, and the ludemic approach has already proven to be a valid and powerful model for evolving games [31].

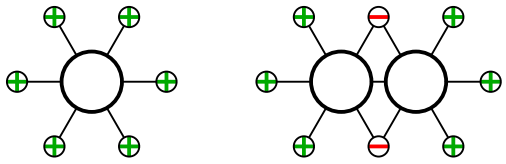


Fig. 11. Patterns that constitute a "make thin groups" strategy.

A. Computational Phylogenetics

Once a genetic framework has been established, *computational phylogenetics* techniques such as those used to create phylogenetic trees mapping the dispersal of human language [40] can be applied. Such techniques allow *ancestral state reconstruction* for estimating the likelihood of given traits occurring in "ancestor" games, and the inference of possible *missing links* in the form of unknown games suggested by the phylogenetic record for which no evidence exists.

Phylogenetic techniques have previously been applied to subsets of Mancala games [41] and Chess-like games [42]. However, phylogenetic analyses of such cultural domains tend to confuse the genotype and phenotype of artefacts, yielding classifications of questionable value based on superficial traits rather than meaningful underlying structures [43]. List *et al.* provide guidelines for correctly casting cultural domains in a biological framework [44].

B. Game Distance

Games do not contain the traces of genetic heritage that biological organisms do; rule sets are typically optimised and superfluous rules stripped out, making their heritage hard to trace. In lieu of a metric for genetic distance, the *ludemic distance* between games will be used, given by the *weighted edit distance* (WED) between ludemic descriptions, i.e. the number of removals, insertions and edits required to convert one into the other, weighted according to the relative importance of each attribute. This is similar in principle to the *Hamming distance* used to quantify the similarity between DNA sequences in bioinformatics [45]. Care must be taken to detect and handle *homologies* [46] that occur when different ludeme structures produce the same behaviour in play.

C. Horizontal Influence Maps

Morrison points out that phylogenetic *networks* may be more suitable than *trees* for modelling the evolution of cultural artefacts [47]. This seems especially relevant for games, which

are more likely to have evolved through distributed *polygenesis* from multiple sources than *monogenesis* from a single common ancestor [48], and in which rules can pass from one to another through *ethnogenesis* (i.e. horizontal gene transfer) rather than classic inheritance. The prevalence of ethnogenesis in the spread of games could warrant the use of *horizontal influence maps* (HIM) [49] (Fig. 12) rather than standard phylogenetic approaches based on vertical gene transfer.

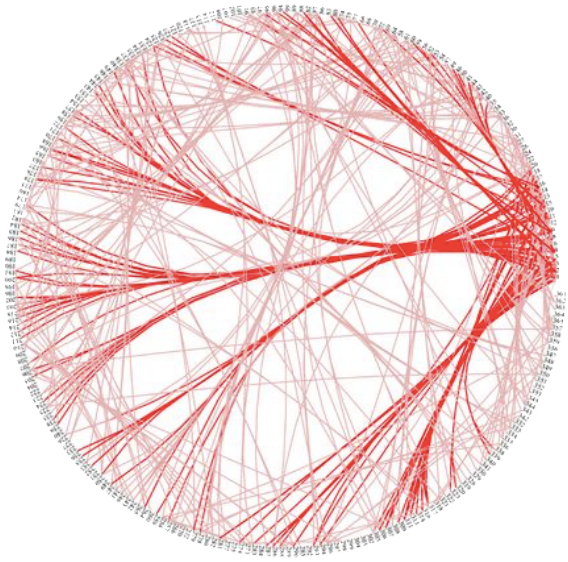


Fig. 12. Horizontal influence map (from [49]).

VII. Cultural Mapping of Games

To facilitate the cultural mapping of games, ludemes and game descriptions will be tagged with relevant metadata:

- *Mathematical:* Ludeme classes will be tagged with the underlying mathematical concepts that they embody.
- *Historical:* Game descriptions will be tagged with details regarding when and where they were played (among other cultural details).

Each game will therefore have a *mathematical profile* based upon its component ludemes and a *historical profile*. The game database will be data-mined for common ludemeplexes that represent important game mechanisms. The associated metadata will be cross-referenced to create *knowledge graphs* that give probabilistic models [50] of the relationships between their geographical, historical and mathematical dimensions.

The cultural location of games will be achieved using a geo-location service such as GeaCron.[7] GeaCron maintains a database of geo-political world maps for every year from 3000BC to the present day, which can be queried to specify which empire, nation, civilisation or culture was dominant at any given geographical location in recorded history.

GeaCron also provides details of known trade routes, expeditions, and other key historical events, for example Fig. 13 shows the Viking route from Denmark to Paris in 845AD. This

[7]http://geacron.com

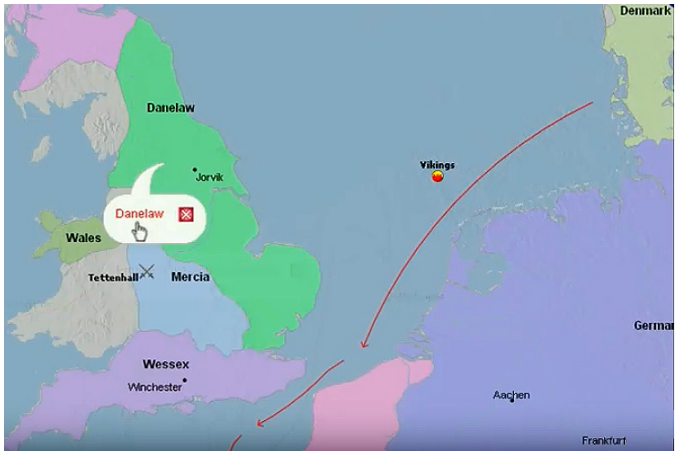


Fig. 13. Viking route from Denmark to Paris in 845AD (image by GeaCron).

provides a mechanism for correlating the spread of games, ludemes and associated mathematical ideas with the spread of human civilisation.

VIII. Digital Archæoludology

With these ideas in mind, I propose a new field of study called *digital archæoludology* (DA), for the analysis and reconstruction of ancient games from incomplete descriptions using modern computational techniques. The aim is to provide tools and methods that might help game historians and researchers better understand traditional games.

Traditional game studies have tended to focus on the authenticity of reconstructions (as cultural artefacts) rather than their actual quality as games. DA seeks to redress this imbalance by searching for plausible reconstructions that maximise both quality and historical authenticity, hopefully leading to better reconstructions, a better understanding of ancient and early games, and a more accurate and complete model of the development of traditional strategy games throughout history.

A. Forensic Game Reconstruction

A key application of DA is the forensic reconstruction of games from partial descriptions, such as the Poprad game shown in Fig. 7. The following equipment is known:

1) Rectangular board with 17×15 or 17×16 square grid.
2) Pieces of two colours.
3) Pieces of two sizes (possibly).

The Ludii system could perform a search of the ludeme space constrained to these requirements, to find plausible rule sets that maximise both game quality and historical authenticity based on what is known about the game, in this case the historical and cultural context of the tomb in which the game was found and its inhabitant. Ludii could fill in the "missing bits" such as finding historically accurate combinations of rules that provide interesting games, what number of pieces provide better results, how they are best arranged to start the game, and so on. The aim is to provide tools for the plausible reconstruction of such missing knowledge, so such tasks no longer seem impossible.

Reviewing the complete set of traditional game reconstructions modelled in the database – to identify implausible cases and optimise them where possible – has the potential to improve our understanding of traditional games. The intention is to create a positive feedback loop in which better reconstructions lead to better historical and cultural mappings, which lead to even better reconstructions, and so on.

IX. Conclusion

Games offer a rich window of insight into our cultural past, but early examples were rarely documented and our understanding of them is incomplete. While there has been considerable historical research into games and their use as tools of cultural analysis, much is based on the interpretation of partial evidence with little mathematical analysis. This project will use modern computational techniques to help fill these gaps in our knowledge empirically, establishing a new field of research called digital archæoludology.

Acknowledgment

This work is part of the Digital Ludeme Project, funded by €2m European Research Council (ERC) Consolidator Grant #771292, conducted at Maastricht University over 2018–23.

References

[1] J. Huizinga, Homo ludens: Proeve eener bepaling van het spel-element der cultuur [Homo ludens: A study of the play-element in culture]. Haarlem: Tjeenk Willink, 1938.
[2] R. Brooks, "Machine learning explained", Robots, AI, and Other Stuff, online report, 2017.
[3] G. N. Yannakakis and J. Togelius, Artificial intelligence and games. Berlin: Springer, 2018.
[4] D. Parlett, The Oxford history of board games. Oxford: Oxford Univ. Press, 1999.
[5] F. Horn and A. de Voogt, "The development and dispersal of lAttaque games", Board Game Studies, 2008, pp. 43–52.
[6] A. de Voogt, "Moving into Micronesia: Checkers and Sorry!", lecture, Board Game Studies Colloquium XX!, Athens, 2018.
[7] A. de Voogt, "Distribution of Mancala board games: A methodological inquiry", Board Game Studies, vol. 2, 1999, pp. 104–114.
www.chessvariants.com
[8] H. J. R. Murray, A history of board-games other than Chess. Oxford: Oxford Univ. Press, 1952.
[9] U. Schädler, "Mancala in Roman Asia Minor?", Board Game Studies, vol. 1, 1998, pp. 10–25.
[10] U. Schädler, "Games, Greek and Roman", R. S. Bagnall, K. Brodersen, C. B. Champion, A. Erskine and S. R. Huebner (eds.), The Encyclopedia of Ancient History. London: Blackwell, 2013, pp. 2841–2844.
[11] W. Crist, A. E. Dunn-Vaturi and A. de Voogt (eds.), Ancient Egyptians at play: Board games across borders. London: Bloomsbury, 2016.
[12] T. Kendall, Passing through the Netherworld: The meaning and play of Senet, an Egyptian funerary game. Belmont: Kirk Game Co., 1978.
[13] E. Duggan, Ancient board games 1: The Royal Game of Ur and Senet, lecture notes, Univ. Suffolk, 2015, unpublished.
[14] I. Finkel, "On the rules for the Royal Game of Ur", Ancient board games in perspective: Papers from the 1990 British Museum Colloquium. London: British Museum Press, 2007, pp. 16–32.
[15] W. Crist, "Games in the ancient Caucasus", lecture, Board Game Studies Colloquium XX!, Athens, 2018.
[16] C. Linnæus, Lachesis Lapponica, J. E. Smith (trans.), vol. II, London, White and Cochrane, 1811.
[17] J. C. Ashton, "Linnaeus's game of Tablut and its relationship to the ancient Viking game Hnefatafl", The Heroic Age: A Journal of Early Medieval Northwestern Europe, vol. 13, 2010, pp. 1526–1867.
[18] H. J. R. Murray, A history of Chess. Oxford: Oxford Univ. Press, 1913.
[19] E. Ertugrul, "Assos Antik Kenti'nde 2300 Yil Önce Üç Taş Oynaniyordu", Arkefili, in Turkish, 7 December 2015.
[20] M. Uberti, The Merels board enigma: With the worldwide census, eBook, Ultimabooks.com, 2015.
[21] F. U. M. Heimann and U. Schädler, "The loop within circular Three Mens Morris", Board Game Studies, vol. 8, 2014, pp. 51–61.
[22] M. Ascher, "Mu Torere: An analysis of a Maori game", Mathematics Magazine, vol. 60, no. 2, 1987, pp. 90–100.
[23] P. D. Straffin, "Position graphs for Pong Hau K'i and Mu Torere", Mathematics Magazine, vol. 68, no. 5, 1995, pp. 382–386.
[24] E. B. Tylor, "On the game of Patolli in ancient Mexico, and its probably Asiatic origin", Journal of the Anthrop. Inst. of Great Britain and Ireland, vol. 8, 1879, pp. 116–131.
[25] A. Caso, "Un antiguo juego Mexicano: El Patolli", Revista de Revistas, vol. 774, March 1925, pp. 40–41.
[26] C. J. Erasmus, "Patolli, Pachisi and the limitation of possibilities", Journal of Southwest Anthropology, vol. 6, 1950, pp. 369–387.
[27] Spectator, "Ancient board game has no parallel in Europe", The Slovak Spectator, 5 Jan 2018.
[28] D. Parlett, "What's a ludeme?", Game & Puzzle Design, vol. 2, no. 2, 2016, pp. 83–86.
[29] C. Browne, Evolutionary game design. Berlin: Springer, 2011.
[30] M. Genesereth, N. Love and B. Pell, "General game playing: Overview of the AAAI competition", AI Magazine, vol. 26, no. 2, 2005, pp. 62–72.
[31] C. Browne, Automatic generation and evaluation of recombination games. PhD thesis, Faculty of Inform. Tech., Queensland Univ. Tech. (QUT), Brisbane, 2009.
[32] C. Browne, "A class grammar for general games", Proc. 9th Internat. Conf. on Computers and Games (CG'2016), Leiden, Springer, LNCS 10068, 2016, pp. 169–184.
[33] C. Browne, *et al.*, "A survey of Monte Carlo tree search methods", IEEE Trans. Comput. Intell. and AI in Games, vol. 4, no. 1, 2012, pp. 1–43.
[34] D. Silver, *et al.*, "Mastering the game of Go with deep neural networks and tree search", Nature, vol. 529, no. 7587, 2016, pp. 484–489.
[35] T. Raiko and J. Peltonen, "Application of UCT search to the connection games of Hex, Y, *Star, and Renkula!", Proc. Finn. Artif. Intell. Conf., Espoo, Finland, 2008, pp. 89–93.
[36] C. Browne, "What can game AI teach us?", ICGA Journal, vol. 37, no. 3, 2014, pp. 171–175.
[37] I. Althöfer, "Computer-aided game inventing", technical report, Univ. Jena, 2003, unpublished.
[38] F. Lantz, A. Isaksen, A. Jaffe, A. Nealen and J. Togelius, "Depth in strategic games", Proc. 31st AAAI Conf. Artif. Intell. (AAAI'17), workshop on What's Next for AI in Games?, San Francisco, AAAI Press, 2017, pp. 1–8.
[39] J. Tromp, "The number of legal Go positions", Proc. 9th Internat. Conf. Computers and Games (CG'2016), Leiden, Springer, LNCS 10068, 2016, pp. 183–190.
[40] S. Greenhill, "Evolution and language: Phylogenetic analyses", International Encyclopedia of the Social & Behavioral Science, 2nd edition, vol. 8, 2015, pp. 370–377.
[41] V. A. Eagle, "On a phylogenetic classification of Mancala games, with some newly recorded games from the 'Southern Silk Road', Yunnan Province, China", Board Games Studies, vol. 1, 1999, pp. 51–68.
[42] A. R. Kraaijeveld, "Origin of Chess – A phylogenetic perspective", Board Games Studies, vol. 3, 2001, pp. 39–50.
[43] D. A. Morrison, "False analogies between anthropology and biology", The Genealogical World of Phylogenetic Networks, 2013, online.
[44] J.-M. List, S. J. Pathmanathan, P. Lopez and E. Bapteste, "Unity and disunity in evolutionary sciences", Biology Direct, vol. 11, no. 39, 2016, pp. 1–17.
[45] C. A. F. Anselmo and A. Pinheiro, "Phylogenetic trees via Hamming distance decomposition tests", Journal of Statist. Comput. and Simul., vol. 82, no. 9, 2012, pp. 1287–1297.
[46] I. Tëmkin and N. Eldredge, "Phylogenetics and material cultural evolution", Current Anthropology, vol. 48, no. 1 2007, pp. 146–154.
[47] D. A. Morrison, "Are phylogenetic patterns the same in anthropology and biology?", bioRxiv:10.1101/ 006486, 2014.
[48] D. Parlett, "Back to square one: Questing the origin of games", The Incompleat Gamester, 2011, online.
[49] S. Valverde, "Visualising the evolution of programming languages", Leonardo, Special Section: Arts, Humanities and Complex Networks 2015, 27 April 2016.
[50] X. Dong and V. A. Eagle, "Knowledge vault: A web-scale approach to probabilistic knowledge fusion", Proc. 20th Internat. Conf. on Knowl. Disc. and Data Mining, New York, ACM Press, 2014, pp. 601–610.

Web-Based Interface for Data Labeling in StarCraft

In-Chang Baek
Dept. of Computer Science and Engineering
Sejong University
Seoul, South Korea
bic4907@gmail.com

Kyung-Joong Kim*
Dept. of Computer Science and Engineering
Sejong University
Seoul, South Korea
kimkj@sejong.ac.kr

***Abstract*—Recently, StarCraft AI has been very actively researched, largely via analysis of human replay data. However, such data are difficult to evaluate visually because they represent information from a limited environment, that of the game client. To solve this problem, we created an environment in which game screens are displayed on the web, allowing game progression to be evaluated at a glance. This allows the performance of more diverse and efficient experiments than conventional human testing. We show that human players label macro decisions (e.g., main force operations) during supervised StarCraft learning using a web-based interface.**

***Index Terms*—human test, StarCraft AI, visualizer, replays**

I. Introduction

StarCraft, which was released in 1998 by Blizzard, is a game of three races. Many experts and optimized strategies are available. A large set of complex replays has attracted AI researchers. For example, the Facebook AI team released 300G of data extracted from about 65,000 games [1].

BWAPI can be used to hack StarCraft clients to obtain the information needed for a game (units, buildings, and statements) and to control certain units via code. BWAPI is also used for StarCraft AI development. StarCraft AI competitions are held every year.

Fig. 1. The screen of StarCraft

Compared to other video games released with the latest forms of learning reinforcement, StarCraft is of the Real Time Strategy (RTS) genre; various environments and conditions that greatly affect game flow are available. Especially, unit combinations and interactions determine the flow of the game and the difficulty of decision-making. In each situation, it is essential to read the flow to predict the future and make the optimal choice. The use of deep/reinforcement learning to make game decisions remains at an early stage of development [2]. In fact, even the existing bots choose StarCraft strategies using a finite state machine (FSM) [3]

Previous research has shown that macro decision-making requires human labeling (e.g., in what direction will the main force move?). The game can turn on the basis of that decision. Also, it is important to predict where the main enemy force is and what actions it will take. The AI model searches every frame for the main force and makes the best possible decision. The real question is as follows: 'Where is the main force and where will it move to in five seconds?' Professional players chose the five-second window. As viewing long games remains difficult, we sought to solve the problem by watching only one scene.

We built a Web-based environment because of the limited functionality of the StarCraft client. As shown in Figure 1, the client views the game screen locally, rendering it difficult for another to evaluate the situation at a glance. Although a mini-map is available, this is not very helpful because the unit type is not shown. In the original StarCraft replay (provided by Blizzard), it is not possible to rewind the game or jump to specific scenes or times. Because of these inconveniences, we created our Web environment. Previously, StarCraft AI performance was evaluated using human players [4]. However, the work was performed offline and time and space constraints were in play. Here, we sought to deal with these issues.

II. A Web-based StarCraft Data Labeling System

Environments such as described here typically provide four functions (TABLE I. A single scene is studied, but it is possible to view past scenes when evaluating changes in successive scenes. Also, the small units in the resized map are now easily identified by color-coding, reducing the time required for evaluation.

We used JSON software to move StarCraft data to the Web environment. StarCraft does not support conversion of replay files to JSON, but BWAPI allows data-writing in any desired format. The data required here are:

- Map, player, and unit information for each frame (position, type, health level, shield level [a Protoss race property], and fog of war in game)

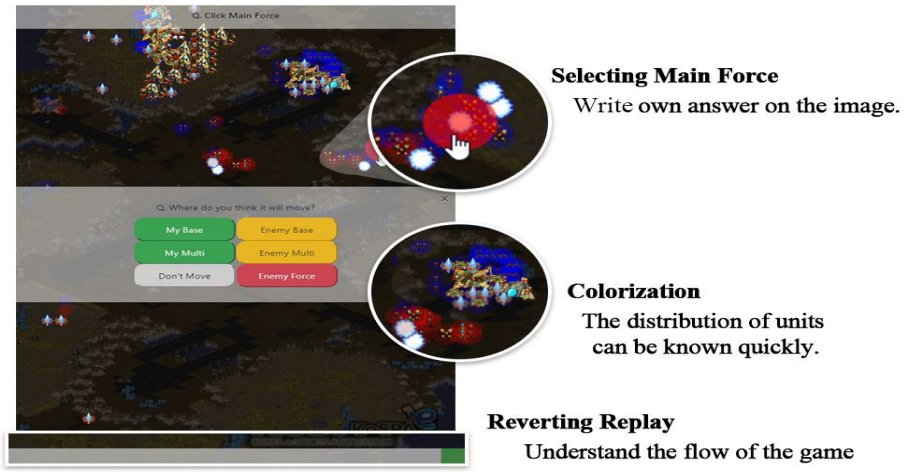


Fig. 2. Web Interface of Environment (https://cilab.sejong.ac.kr/sc/evalution)

This information defines only one scene and identifies only the main force, but it shows a second scene about 3 minutes into the future. The method is implemented in Python and extracts and compresses only the required data when the user solves the problem employing the JSON file. The data created are distributed via a Web socket. The server submits the problem to the user in real time after considering the submission situation. The interface creates an HTML5 canvas of the problem dataset. We constructed the game screen using available game data and implemented a function sending human answers to the server.

TABLE I. FUNCTIONS OF ENVIRONMENT

Function	***Description***
Rewind	Watch future scences like a video.
Colorization	Learn the distributions of units (allies, enemies).
Coordinate selection	Mark the position on the image.
Option selection	Additional questions can be posted.
Time measurement	Measuring the time it takes to solve a problem.

The system collects data from human players in the following format:

- Replay name, human tester ID, frame count (scene number), elapsed time, main force position, and decision.

The file name and frame count are used both to match the scene in question and for AI prediction. Performance is measured by the differences between the co-ordinates selected by the human and the AI, the main force-detection algorithm. If the positional difference is within a given error range, the behavioral performance can be scored.

III. CONCLUSION AND FUTURE WORKS

Unlike existing human tests [4], the human evaluator provides customized functions by moving from the limited environment of a game client to a Web-based environment. If the game client lacks the necessary interface for experimentation, or if it is felt that use of such an interface would waste time, our approach improves productivity (saving time and expanding knowledge). For example, not only is it possible to evaluate a given scene it is also possible to add an item and evaluate the previous scene, allowing interpretation of the experiment.

Our work was based on StarCraft I, but it is extensible to various interfaces and problem types as required. Our interfaces can be applied to study other RTS games where flow is important, including Warcraft and StarCraft II.

ACKNOWLEDGEMENTS

This research was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT & Future Planning (grant no. 2017R1A2B4002164).
*: corresponding author

REFERENCES

[1] Z. Lin, G. Jonas, K. Vasil, and G. Synnaeve, “STARDATA: A StarCraft AI Research Dataset,” AIIDE 2017.

[2] I. S. Oh and K. J. Kim, "Testing reliability of replay-based imitation for StarCraft," 2015 IEEE Conference on Computational Intelligence and Games (CIG), Tainan, 2015, pp. 536-537.

[3] B. G. Weber, P. Mawhorter, M. Mateas and A. Jhala, "Reactive planning idioms for multi-scale game AI," Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games, Dublin, 2010, pp. 115-122.

[4] M. J. Kim, K. J. Kim, S. Kim and A. K. Dey, "Performance Evaluation Gaps in a Real-Time Strategy Game Between Human and Artificial Intelligence Players," in IEEE Access, vol. 6, pp. 13575-13586, 2018.

Author Index

Keyword Index