MCTS in OpenTTD[1]

Geert Konijnendijk

June 2015

Abstract

In this thesis an agent for the Construction and Management Simulation game OpenTTD is presented, called MCTrAInS. OpenTTD is a game in which a player manages a transport company and constructs a transportation network using various modes of transport. MCTrAInS makes use of the Monte-Carlo Tree Search (MCTS) algorithm for the planning of routes. A model of the game is discussed that enables the use of MCTS in OpenTTD. Two search enhancements to the MCTS algorithm are implemented in MCTrAInS, namely Progressive History and ϵ-greedy playouts. Progressive History seems to improve the performance of the agent, however ϵ-greedy playouts did not. MCTrAInS is compared to two pre-existing agents for OpenTTD, trAInS and AdmiralAI. MCTrAInS is an improvement over trAInS, but AdmiralAI outperforms MCTrAInS.

Keywords: Monte-Carlo Tree Search, OpenTTD

1 Introduction

Transportation and travel are both major parts of modern society. Every day trains, ships, trucks, busses and aeroplanes transport people and goods all over the world. For this transportation to operate, the necessary infrastructure has to be constructed and the routes and time schedules of the vehicles have to be planned.

A genre of video games called Construction and Management Simulation (CMS) [13] lets a player run a company or other entity using construction and management tools. The goal is usually to optimize a process in the entity the player is controlling. The company a player is managing could be a transportation company, but there are also different examples. For instance, a well-known CMS game is SimCity in which the player builds a city and manages aspects like taxes [1].

CMS games mostly try to model a certain level of realism, as is the case in SimCity, where a player can even build and manage real cities like Tokyo. However, a perfect simulation of real life is not possible and mostly not fun enough to be a video game.

Many CMS games employ Artificial Intelligence (AI) tasked to control the simulation or agents in the simulation, for example the people walking through a city in newer versions of SimCity.

Another task for AI in the CMS genre is to actually play the game, so a human player will have a competent opponent when no other human player is at hand. Creating such an AI agent is a challenging problem because the agent playing the game should be able to perform all the construction and management tasks at preferably at least the same level as a human player.

A CMS game which, like SimCity, is also well known is OpenTTD (Open Transport Tycoon Deluxe). In OpenTTD a player manages a transportation company and builds transport routes using multiple modes of transport. SimCity is a mostly single-player experience, however multiple players can play together on a single map in OpenTTD. This makes OpenTTD a logical choice to develop AI agents for. There are many AI agents available for OpenTTD, using a host of different AI techniques. Some examples are PathZilla[2], an agent using graph theory for network planning, trAIns, which uses a modified version of the A* algorithm [7] for railroad construction and SPRING [20], which investigated using a classic planner for management tasks in OpenTTD.

Most parts of management in OpenTTD agents are handled by rule-based approaches by many of the existing agents. They work relatively well, however it is expected that by using more sophisticated AI techniques like search algorithms the existing agents can be improved.

Only few simulation games have employed search algorithms, like Monte-Carlo Tree Search (MCTS) [4, 8] in the agents playing their game. An example of a game that does use MCTS in its agent is the hybrid CMS and war game Total War: Rome II [2]. This agent uses MCTS to assign resources to specific tasks, therefore tak-

[1] This thesis has been prepared in partial fulfillment of the requirements for the Degree of Bachelor of Science in Knowledge Engineering, Maastricht University, supervisor: Dr. Mark Winands.

[2] `http://www.tt-forums.net/viewtopic.php?f=37&t=38645`

ing care of some of the management tasks the agent has to perform.

1.1 Problem Statement and Research Questions

To find out if MCTS is applicable in other CMS games, the problem statement "*In what parts of management and construction in a CMS game can MCTS be successfully applied?*" is investigated. In this thesis the Monte-Carlo Tree Search algorithm is applied to the CMS game OpenTTD. MCTS was selected over other search algorithms like A* and minimax for the following reasons. In A* an admissible heuristic, i.e. a heuristic that never overestimates cost, needs to be used, however such a heuristic is hard to design for OpenTTD. Minimax was not selected for a similar reason because it requires an evaluation function, which is also hard to design. The following two research questions aid in investigating the problem statement.

1. How can OpenTTD be fit into the framework required by MCTS?

   In MCTS a cloneable state has to exist to run the required playouts, furthermore actions have to be defined that will lead from one state to the next. Also, the value of terminal states has to be determined at the end of playouts.

2. Can an agent utilizing MCTS outperform already existing agents?

   Members of the OpenTTD community have created multiple agents for playing the game against human players or other AI agents. The agent utilizing MCTS created for this thesis, called MCTrAInS, is compared to some of these agents.

1.2 Outline

The structure of this thesis is as follows. Section 2 introduces the CMS game OpenTTD. Section 3 discusses the MCTS algorithm and some of its enhancements that are implemented in MCTrAInS. Furthermore it explains the state and actions used in the MCTrAInS agent. Section 4 describes how the MCTS algorithm is integrated into OpenTTD to create a functioning agent. Thereafter the experimental setup and results are discussed in Section 5. In Section 6 conclusions are drawn and discussed. Finally, Section 7 outlines possible future research.

2 OpenTTD

OpenTTD is an open-source CMS game in which one or more players try to create the most efficient transportation network possible [11]. It is based on the game Transport Tycoon Deluxe which was created by Chris Sawyer,

Figure 1: Screenshot of OpenTTD

the creator of other CMS games like Roller Coaster Tycoon. Transport Tycoon Deluxe was released in 1995 and published by Micropose [15]. The development of OpenTTD started in 2004 and it is still being developed actively.

OpenTTD is usually played on a randomly generated map with a number of industries and cities divided over it, all producing and accepting different kinds of cargo in varying amounts. Maps can also be created by hand, although this is less common than random generation. A player can use trucks, busses, trains, ships and aeroplanes to deliver passengers and cargo from one city or industry to another. OpenTTD also has chains of cargo. For example a mine produces iron ore, which can be transported to a steel mill, which produces steel to be transported to a factory producing goods.

A considerable improvement of OpenTTD over the original is the addition of a multiplayer mode. Players can play either cooperatively, by controlling one company with multiple people or competitively, by controlling one company per person. When there is a lack of other human players, AI agents can play against other players in both single-player and multiplayer modes.

Since the game is a CMS game, it resembles the management and construction of transport routes in real life. A player has to construct infrastructure like roads in the case of trucks or buses and tracks, signals and stations in the case of trains. Vehicles have to be bought and their orders planned. They should be maintained to prevent them from breaking down too often and when they become too old they should be sold and replaced by newer models. If the player has planned a transport route carefully it will create revenue to spend on other routes.

When classified by Russel's and Norvig's [14] environment properties the game is fully observable, either single or multi-agent, non-deterministic, dynamic (real-time) and discrete. The agent created in this thesis assumes the single-agent case by ignoring the opponent. This is done because there is little interaction be-

tween multiple players, they are usually only working on their own transportation network. The game is non-deterministic because some events like vehicles breaking down happen randomly, but in general most events are deterministic.

3 MCTS Route Planner

To be able to play OpenTTD the MCTrAInS agent was developed. This agent uses the MCTS algorithm for route planning. Firstly, MCTS is discussed in Subsection 3.1 and afterwards the model used for simulating is discussed in Subsection 3.2.

3.1 MCTS

MCTS is a best-first search algorithm with its main uses in AI for games [4, 8]. In MCTS each node represents a state of the game and each edge represents an action which leads from one state to the next when executed. The algorithm explores the state-space in a random fashion, where further in the execution the algorithm is directed by its previous results, striking a balance between exploration and exploitation. MCTS is based on four steps, which are executed repeatedly until a stopping condition is reached, mostly this condition is a set time limit [3]. The four steps are explained below. These steps are illustrated in Figure 2.

Selection

In the selection step child nodes are selected repeatedly until a state which is not part of the tree is reached. In the MCTrAInS agent UCT (Upper Confidence Bounds applied to Trees) [8] combined with Progressive History [9] are used as a selection strategy.

The UCT calculation is shown in Formula 1. This formula determines if the algorithm explores or exploits more. It is applied to all child nodes i of parent p. Usually, the child with the highest value of the UCT formula is selected. In the formula $\bar{x}_i$ is the value of node i, n_p is the number of times p was visited and n_i is the number of times i was visited. C is a constant which should be tuned experimentally.

$$\bar{x}_i + C \times \sqrt{\frac{\ln(n_p)}{n_i}} \tag{1}$$

UCT alone does not perform reliably when nodes only have a small number of visits. A way to circumvent this problem is to apply the Progressive History technique in the selection strategy. Progressive History is selected as an MCTS enhancement since it has proven to be a significant improvement over standard UCT [9, 10].

Progressive History assumes that actions which had a good result in one situation will likely have a good result when played in another. To keep track of how successful actions are, actions previously executed and their average resulting score at the end of a playout are stored. This history is then used to prefer selection of states which are the result of better actions over those which are the result of worse actions. The Progressive History calculation is shown in Formula 2. In this formula $\bar{x}_a$ is the average score of all games in which action a was executed. W is a parameter determining the influence of Progressive History which should be tuned experimentally. All other parameters are the same as described above for Formula 1.

$$\bar{x}_a \times \frac{W}{(1-\bar{x}_i)n_i + 1} \tag{2}$$

When the two formulas are combined, Formula 3 is the result. Using this formula, starting from the root, the child node with the highest v_i is selected until a leaf node is reached.

$$v_i \leftarrow \bar{x}_i + C \times \sqrt{\frac{\ln(n_p)}{n_i}} + \bar{x}_a \times \frac{W}{(1-\bar{x}_i)n_i + 1} \tag{3}$$

Playout

In the playout step, one game is played until the end. This is performed by playing random moves until a terminal state is reached. The simulation strategy can also be enhanced by selecting moves other than purely random ones to make playouts more representative of actual playouts, thereby being able to play better. Such an enhancement called ϵ-greedy playouts [16] is implemented in MCTrAInS. The ϵ-greedy playouts enhancement entails that every time an action has to be selected, a heuristic is used to choose the best possible action. This action is selected with probability $1 - \epsilon$. With probability ϵ a random action is selected. The heuristic used in MCTrAInS is to choose the action that will build the route with the highest monthly income as the best action. This action will therefore be played in the playouts with a probability of $1 - \epsilon$.

Expansion

In the expansion step, new children of the node selected in the previous step are added to the tree. In MCTS all children of a node can be added in one go, or only a subset of the children can be added. In MCTrAInS only one child is expanded per iteration.

Backpropagation

During backpropagation the score of each node from the currently selected leaf node to the root node is updated and its visit count incremented. The score in MCTrAInS is calculated by using the sigmoid function in Formula 4. In this formula s is the score and λ is the estimated income in pound sterling.

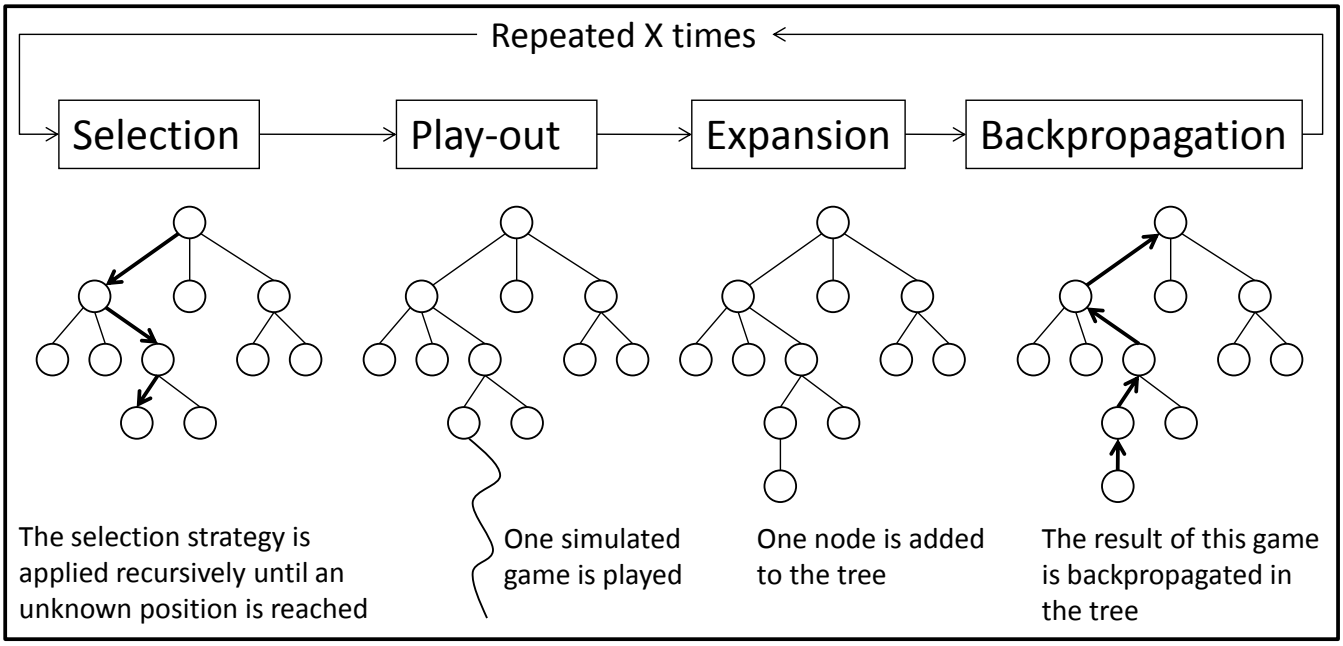

Figure 2: Steps of MCTS

$$s \leftarrow \frac{1}{1 + e^{10^{-6}(-\lambda + 800000)}} \tag{4}$$

3.2 Model

In its default state OpenTTD is not well suited for usage by tree search algorithms like minimax and MCTS since it does not contain a forward model. Given an action, such a model should be able to calculate the following state. This section outlines the major design decisions of a model for OpenTTD and describes its implementation.

A model for OpenTTD should strike a balance between realism and simplicity. When the model does not imitate the internal workings of OpenTTD well enough MCTS will not make adequate decisions. However, when the model takes too many features of OpenTTD into account it could become too slow to simulate with, also causing MCTS to make poor decisions because of a lack of iterations. Construction in OpenTTD is a process with many decisions to be made, like where to place railroads, on what piece of track to place signs and where and what kind of bridges to build. Considering all these possibilities will result in a branching factor that is most likely too high to be feasible for MCTS. Therefore it was decided to implement MCTS and its model only for the management part of the agent. MTCS combined with the model make decisions about relatively high-level management tasks like building, upgrading and demolishing railroad routes.

In the model abstract representations of objects in the OpenTTD game world are stored. These include industries, cities and routes. The MCTRAINS agent only considers railroad routes since they are the mode of transport with most features, like signals, waypoints and trains of varying length. They are usually the most efficient kind of transport too. Furthermore simple data like the current amount of money available and the company value is stored in the model. In the following paragraphs the representation of industries, cities and routes is elaborated on.

Both cities and industries are represented in a similar fashion in the model. They store their location on the map and an estimation of their cargo production. In addition to the previously mentioned attributes industries also store the types of cargo they produce and the types of cargo they accept, these are always passengers in the case of cities.

In the model routes represent the connections between two industries or cities. Therefore the combination of routes, cities and industries can be seen as a graph with each node containing an industry or city and each edge representing a route. An example of such a graph can be seen in Figure 3. Note that all routes are modelled as a separate edge, but in the game map one route might be using railroad track created for a different route. This is shown as the dotted line in Figure 3. Routes store their source and destination industry, the type of vehicles currently operating on the route, the type of cargo transported over the route, the estimated cost for building the route and their estimated income. When an action to build a route is generated in the playout step of MCTS,

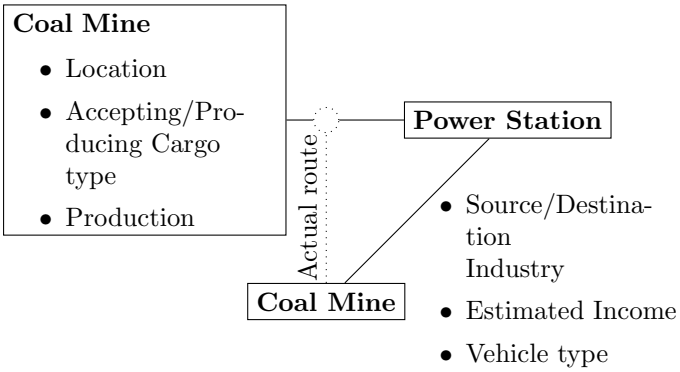

Figure 3: Routes Graph

Figure 4: Route constructed by trAIns

discussed in Subsection 3.1, only routes with a Manhattan distance between 20 and 100 from their source to their destination are considered because routes which are any shorter or longer are usually not profitable.

There are three operations that can be executed on the model. The first is building a route which will result in building tracks in the actual game and populating them with vehicles. The second one is removing a route, which will sell all vehicles operating on that route. The third is upgrading a route, which will sell the current vehicles on that route and replace them with better ones.

4 Agent Architecture

The MCTS algorithm and model discussed in Section 3 alone do not suffice for creating a complete agent for playing OpenTTD. Subsection 4.1 discusses how the MCTS algorithm and the model are integrated into OpenTTD. The construction of railroad routes is discussed in Subsection 4.2.

4.1 MCTS Integration

The main features of OpenTTD are all written in the C++ programming language. The game features several Application Programming Interfaces (API), accessible using the Squirrel scripting language [5] to allow for easy additions to the game. Developers are able to write agents to play OpenTTD using the NoAI API.

Since MCTS requires a large number of iterations to perform adequately, a scripting language would not be suitable for its implementation. Therefore MCTS, discussed in Subsection 3.1 and the model, discussed in Subsection 3.2 were implemented in C++.

To allow agents written in Squirrel to make use of MCTS, methods were added to OpenTTD's API. These methods make the creation of a new model instance, which is filled with information from the current game state, possible and allow for starting a MCTS run and retrieving the results of a run.

OpenTTD is a real-time game, therefore the environment can change while an agent is deliberating. The MCTrainS agent will not take any of these changes into account and will execute a chosen action using the information it had available when it began planning. When it is not possible to execute an action anymore it will simply stop the execution and try to revert what was already affected.

4.2 Route Building

To execute the actions found by the MCTS implementation, the trAIns agent [12] was adapted resulting in the MCTrAInS agent. The trAIns agent was chosen since it is able to build efficient railroad routes. The agent cannot use any other types of transport, however trains are usually the most efficient type of transport in OpenTTD, but also the type of transport that is most difficult to manage and construct. Furthermore trAIns has functions for railroad construction that can be easily decoupled from the management components. The trAIns agent was created by Luis Henrique Oliveira Rios and Luiz Chaimowicz. The main focus of this agent is building efficient railroad routes, but it is lacking in management tasks. Therefore the part of trAIns managing which routes to build, upgrade or destruct is removed and replaced by the MCTS route planner discussed in Subsection 4.1. The parts of trAIns building the railroad routes are left in place. This results in MCTS making the macro-decisions while the trAIns agent makes the micro-decisions.

trAIns uses a number of techniques for building railroads, these are discussed in the remainder of this section. To create a railroad route, tracks first have to be built. The agent will always build double railroad tracks, these allow trains on one track to travel from the source to the destination and the other way on the other track. This allows large numbers of trains to use one route and prevents collisions. These double railroad tracks are im-

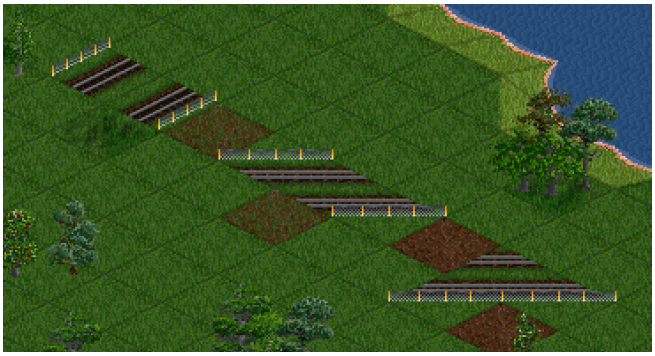

Figure 5: Track parts

plemented by using pre-determined combinations of railroad track as atomic parts for building a route. Figure 5 shows the different combinations of track used as parts. By combining these parts a double railroad can be built easily.

In addition to simply building routes from one source to one destination, the agent also has the possibility to connect a new route to an existing one. This enables the transportation of more goods while reducing the build cost of a route. Connecting routes is implemented using junctions. Junctions are pre-constructed parts, each of them being able to connect to the double railroad parts.

To determine the path tracks are built on, an adaptation of the NoAI A* algorithm [7] is used. A* is an algorithm for finding a shortest path between two nodes in a graph. It using a heuristic function $h(x)$ to estimate the distance to the goal and uses this information to expand fewer nodes while retaining optimality if the heuristic is admissible. Since railroad parts can be placed diagonally the $h(x)$ used in this implementation is Euclidian distance. Another notable change from the original implementation is that in case of a tied $h(x)$ the node chosen is the one minimizing the number of direction changes, increasing the speed a train can travel over the tracks.

5 Experiments

In order to tune the performance of the MCTS algorithm used in MCTrAInS and to compare it to other already existing agents, multiple experiments were designed and run. The setup of these experiments is discussed in Subsection 5.1 and the acquired results are discussed in Subsection 5.2.

5.1 Experimental Setup

To be able to run experiments efficiently, a customized version of OpenTTD without GUI and networking capabilities was used to be able to simulate as fast as possible.

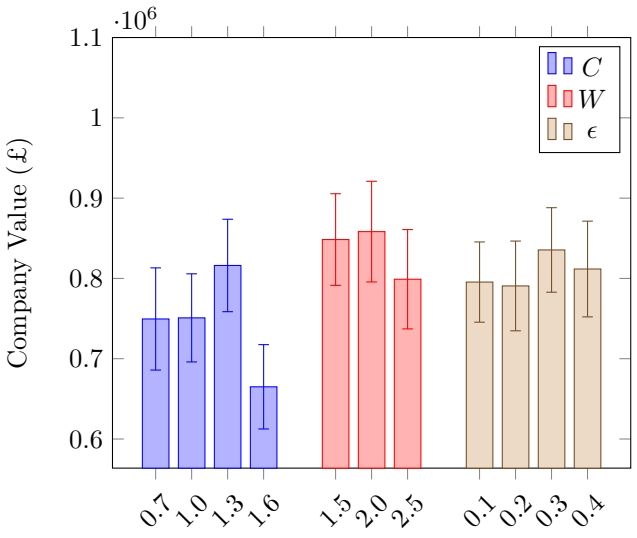

Figure 6: Parameter Tuning

The MCTrAInS agent was compared to two other already existing agents, namely trAIns and AdmiralAI.

The trAIns agent can only build train routes. It features rule-based management components and assigns scores to different buildable routes and builds the route with the best score. The construction components of this agent are discussed in Subsection 4.2.

AdmiralAI[3] is an agent that tries to use as much of the features of OpenTTD available as possible. Therefore it also uses multiple vehicle types, as opposed to MCTrAInS and trAIns. AdmiralAI uses a rule-based approach to the management of routes. Like trAIns it assigns a score to routes and builds the one that it assumes best. The latest version of AdmiralAI was released in 2011, however it is still one of the best known AI agents for OpenTTD.

All experiments were run on a 256×256 map with flat terrain type. As usual in OpenTTD, the maps were randomly generated. Agents were given a maximum loan of £300,000. The experiments all lasted five in-game years, where the MCTS of the MCTrAInS agent was run each in-game month. All experiments were started in the in-game year 1950. In all experiments the company value in pounds sterling was measured. Company value was measured because in OpenTTD it functions as a scoring mechanism for performance of a company. The measurements of the company value were done each quarter, equivalent to 3 in-game months.

In each run the MCTS algorithm was given 10 seconds of running time, or enough time to do 50,000 iterations. This extra time was usually only used in the first run of MCTS because the number of available actions

[3] http://dev.openttdcoop.org/projects/ai-admiralai

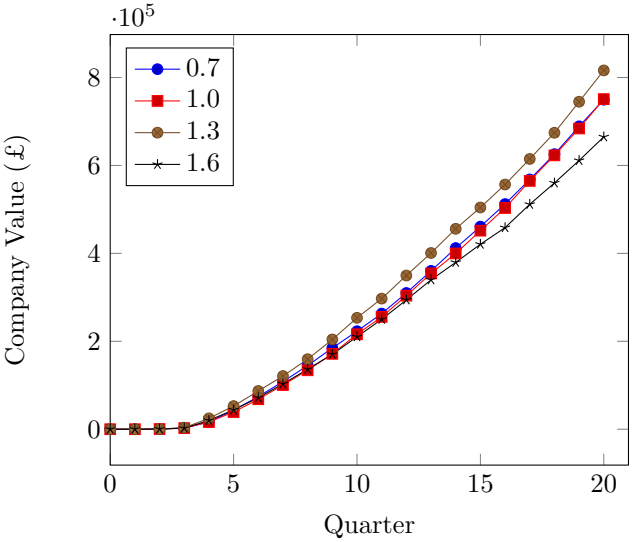

Figure 7: C parameter tuning

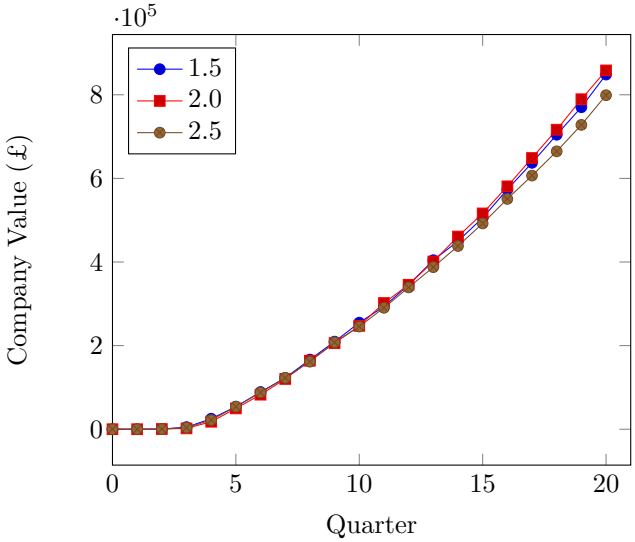

Figure 8: W parameter tuning

is large due to the amount of money available and the lack of routes that have been built. Because OpenTTD is not slowed down by a GUI or networking in these experiments, one game without the MCTS algorithm running finishes fast. Therefore the game is paused during MCTS simulations, effectively making the game turn-based. When the game runs at normal speed and is not paused during the running time of MCTS, only a few days will pass, so there will be no large changes in the game world.

Most experiments were run on computers containing two AMD Dual-Core Opteron F 2216 processors running at 2.4GHz and containing 8GB of RAM. However, the experiments were single-threaded and therefore only used one core. The remaining portion of experiments was run on computers containing two Intel Quad-Core Xeon 5355 processors running at 2.66GHz and containing 8GB of RAM. Using this setup an average run took 19 minutes to complete.

Five different experiments were run. The first three involve tuning the constants C, used in UCT, W, used in the Progressive History heuristic and ϵ, used in the ϵ-greedy playouts. The last two experiments are a comparison between AdmiralAI an MCTRAInS and between TRAIns and MCTRAInS.

5.2 Results

Parameter Tuning

The C, W and ϵ parameters were tuned as follows. First Progressive History and ϵ-greedy playouts were disabled. Then C was tested with different values. Afterwards the best value for C was kept and W was tuned. Then the best W was kept and ϵ was tuned. Figure 6 shows the mean and 95% confidence interval of the company values achieved by the end of the five years for the three experiments at all values tested. Figures 7, 8 and 9 show the company values per quarter for the C, W and ϵ parameters respectively. All values are the result of 100 runs.

When considering the values for the C parameter in Figure 7, a value of 1.3 seems to achieve the best mean in all quarters, therefore this value was selected for use in further experiments. When taking a more exploitative approach, represented by the values 0.7 and 1.0 the mean appears to drop compared to $C = 1.3$ and the mean drops even more when $C = 1.6$, the more explorative approach.

If Progressive History is enabled, only a small improvement in mean over plain UCT is observed when W is set to 1.5 and 2.0. This improvement can be observed in the last five quarters of the running time of the experiment. When the history heuristic is given more influence, which is the case when $W = 2.5$ then the performance drops below that of plain UCT with $C = 1.3$. Possibly the assumption that actions resulting in a good result in one situation will also do so in another does not hold often enough to be able to give Progressive History a large influence, i.e. set W to a high value. Doing so lowers the resulting company value. A value of $W = 2.0$ was used in the following experiments.

Opposite to Progressive History, enabling ϵ-greedy playouts did not increase the mean company value on any of the values of ϵ tested. This might be caused by the heuristic for selecting the best move, discussed in Subsection 3.1, not performing adequately. Because ϵ-greedy playouts do not provide any significant improvement over using only UCT combined with Progressive History, it was left disabled for further experiments.

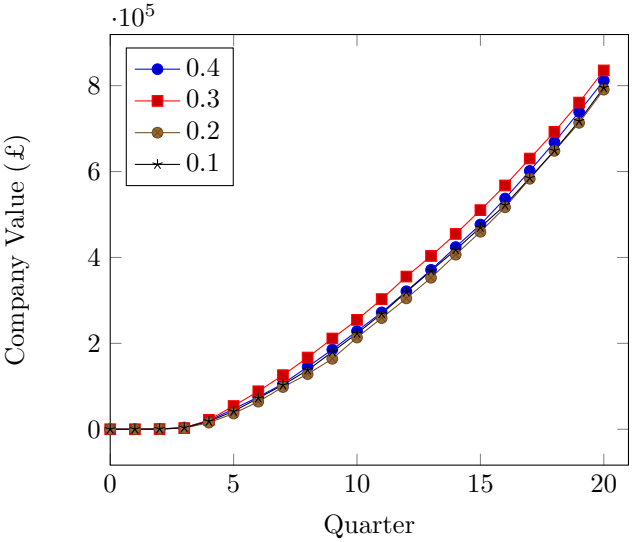

Figure 9: ϵ parameter tuning

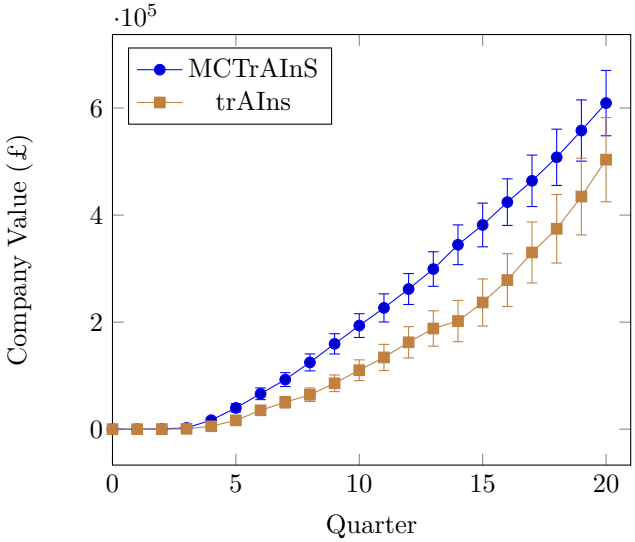

Figure 10: MCTrAInS against trAIns

To test for significance of the measurements an ANOVA test with a significance level of 5% was executed on the company values of the last quarter. The resulting p-value is 2.1230×10^{-4}, meaning that the null hypothesis can be rejected and at least two of the means are significantly different. When followed up by a post-hoc test using Tukey's Honestly Significant Difference Procedure [18] the pairs of measurements differing significantly are $C = 1.6$ compared to $C = 1.3$, $W = 1.5$, $W = 2.0$, $\epsilon = 0.3$ and $\epsilon = 0.4$. This suggest that Progressive History is not a significant improvement over plain UCT and ϵ-greedy playouts are not significantly worse than UCT combined with Progressive History. However, further experiments with other values for W and ϵ are needed to confirm this.

Agent Comparisons

Using the values for the parameters determined in the previous experiments, the MCTrAInS agent was compared to the AdmiralAI agent, a well-known agent for OpenTTD and the trAIns agent, of which MCTrAInS re-uses the infrastructure construction components. Two experiments were executed, in these experiments MCTrAInS competes against the two other agents separately, but in each experiment the two agents play on the same randomly generated maps. All results are based on 100 runs.

Figure 10 shows the mean company value and 95% confidence interval of the MCTrAInS and trAIns agents per in-game quarter over the five years of playing time. The MCTrAInS agent maintains a higher mean company value throughout all five years, outperforming trAIns. This is also reflected in the win rate, MCTrAInS won 72% of the games played and trAIns won 28%. Welch's t-test [19], with a significance level of 5% was used to confirm that the mean company values of both agents in the last quarter of the five in-game years are significantly different. The test resulted in a p-value of 0.0398, so the null hypothesis can be rejected and the two means are significantly different.

Comparing MCTrAInS and AdmiralAI results in a different outcome for MCTrAInS compared to the previous experiment, this can be seen in Figure 11. Not only is the mean company value achieved by AdmiralAI consistently higher than that achieved by MCTrAInS, the mean company value achieved by MCTrAInS is also lower than in any previous experiments. AdmiralAI has an advantage over MCTrAInS because it can use all available modes of transport. Trains are usually more efficient than other modes of transport, but in the cases they are not, AdmiralAI can choose a better one. Another possible cause could be that the AdmiralAI agent is building routes which could be profitable for MCTrAInS before MCTrAInS is able to build them. The win rate also shows that AdmiralAI performs best because it won 99% of the games played. Again, Welch's t-test with a significance level of 5% was applied to the company values of the last quarter. This resulted in a p-value of 9.2004×10^{-36}. Therefore the null hypothesis can be rejected and the two means are significantly different.

6 Conclusion

In this thesis the MCTS algorithm has been applied as part of an agent for playing the CMS game OpenTTD. A model for the management components of OpenTTD has been designed for use with the MCTS algorithm,

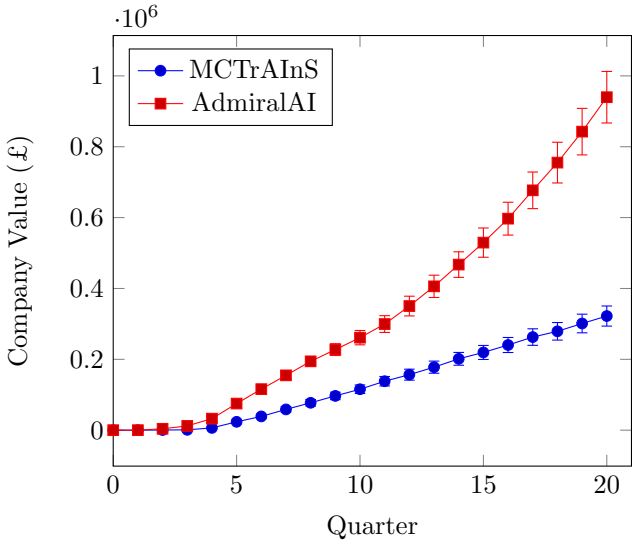

Figure 11: MCTrAInS against AdmiralAI

abstracting the OpenTTD state and game logic. MCTS was combined with several search enhancements in order to improve performance. Progressive History seems to have a positive effect on performance, while ϵ-greedy playouts perform equal or less compared to only Progressive History.

It has been shown that MCTS is applicable in management components of OpenTTD. The implementation of an agent for OpenTTD presented in this thesis, MCTrAInS, outperforms the trAIns agent when comparing company values. It did not outperform AdmiralAI, it is likely that the construction components of the AdmiralAI agent perform better than those of the trAIns agent, upon which MCTrAInS is based.

7 Future Research

OpenTTD has a large number of game features, only a few of which have been used by the MCTrAInS agent. Possible future research is therefore to implement more of these features, like using trucks, boats and aeroplanes in the agent. This will enable the agent to use the optimal mode of transport for a given route, instead of always building a train route. A possible way to achieve this is by integrating the MCTS algorithm into an agent other than trAIns that makes use of more of OpenTTD's features. Furthermore, the model could be made more detailed, for instance by incorporating the acceleration model of vehicles. However, a possible downside could be that the playout will slow down. A balance between detail in the model and speed of the model should be maintained.

In further research other improvements for MCTS could be investigated. For example, an improvement that could replace the ϵ-greedy playouts is MAST [6]. Probably the ϵ-greedy playouts did not perform well because the heuristic domain knowledge used was not adequate. MAST uses similar knowledge as the Progressive History technique and could therefore be a better heuristic for the playouts. An extension on MAST that can also be implemented in future research uses N-Grams instead of the single actions used by MAST. This N-Grams Selection Technique (NST) [17] could be advantageous in OpenTTD since it is likely that there are a few good actions, for example three possible routes that can be quite profitable, which are useful to execute in sequence. NST would exploit these sequences and could lead to improved results of the algorithm.

References

[1] Albert, E. (2001). SimCity Classic History and Review. `http://web.stanford.edu/group/htgg/cgi-bin/drupal/sites/default/files2/ealbert_2001_1.pdf`.

[2] Champandard, A.J. (2014). Monte-Carlo Tree Search in TOTAL WAR: ROME IIs Campaign AI. `http://aigamedev.com/open/coverage/mcts-rome-ii/`.

[3] Chaslot, G.M.J-B., Winands, M.H.M., Herik, H.J. van den, Uiterwijk, J.W.H.M., and Bouzy, B. (2008). Progressive strategies for monte-carlo tree search. *New Mathematics and Natural Computation*, Vol. 4, No. 3, pp. 343–357.

[4] Coulom, R. (2007). Efficient selectivity and backup operators in Monte-Carlo tree search. *Computers and Games (CG 2006)* (eds. P. Ciancarini and H.J. van den Herik), Vol. 4630 of *LNCS*, pp. 72–83, Springer-Verlag, Berlin, Germany.

[5] Demichelis, A. (2015). `http://squirrel-lang.org/`.

[6] Finsson, H. and Björnsson, Y. (2008). Simulation-Based Approach to General Game Playing. *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence*, pp. 259–264. AAAI Press, Menlo Park, California.

[7] Hart, P.E., Nilsson, N.J., and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions of Systems Science and Cybernetics*, Vol. 4, No. 2, pp. 100–107.

[8] Kocsis, L. and Szepesvári, C. (2006). Bandit Based Monte-Carlo Planning. *Proceedings of the 17th European Conference on Machine Learning*, ECML'06, pp. 282–293, Springer-Verlag, Berlin, Germany.

[9] Nijssen, J.A.M. and Winands, M.H.M (2011). Enhancements for Multi-Player Monte-Carlo Tree Search. *Computers and Games (CG 2010)* (eds. H.J. van den Herik, H. Iida, and A. Plaat), Vol. 6515 of *LNCS*, pp. 238–249, Springer-Verlag, Berlin, Germany.

[10] Nijssen, J.A.M. (2013). *Monte-Carlo Tree Search for Multi-Player Games*. Ph.D. thesis, Maastricht University, The Netherlands.

[11] OpenTTD Team (2014). `https://www.openttd.org/en/about`.

[12] Rios, L.H.O. and Chaimowicz, L. (2009). trAIns: An Artificial Inteligence for OpenTTD. *Simposio Brasileiro de Jogos e Entretenimento Digital*, pp. 52–63. Rio de Janeiro, Brazil, eighth edition.

[13] Rollings, A. and Adams, E. (2003). *Andrew Rollings and Ernest Adams on Game Design*. New Riders.

[14] Russel, S.J. and Norvig, P. (2010). *Artificial Intelligence. A Modern Approach*. Pearson Education, Upper Saddle River, NJ, third edition.

[15] Sawyer, C. (2005). `http://www.chrissawyergames.com/info.htm`.

[16] Sturtevant, N.R. (2008). An Analysis of UCT in Multi-player Games. *Computers and Games (CG 2008)* (eds. H.J. van den Herik, X. Xu, Z. Ma, and M.H.M. Winands), Vol. 5131 of *LNCS*, pp. 37–49, Springer-Verlag, Berlin, Germany.

[17] Tak, M.J.W., Winands, M.H.M, and Björnsson, Y. (2012). N-grams and the last-good-reply policy applied in general game playing. *IEEE Transactions on Computational Intelligence and AI in Games*, Vol. 4, No. 2, pp. 73–83.

[18] Tukey, J.W. (1949). Comparing individual means in the analysis of variance. *Biometrics*, Vol. 5, No. 2, pp. 99–114.

[19] Welch, B.L. (1947). The generalization of 'student's' problem when several different population variances are involved. *Biometrika*, Vol. 34, No. 1-2, pp. 28–35.

[20] Wisniewski, M. (2011). Artificial Intelligence for the OpenTTD Game. M.Sc. thesis, Technical University of Denmark, Lyngby, Denmark.