

Monte-Carlo Evaluations in Ms. Pac-Man

I.M.M. Brauers

April 27, 2011

Abstract

This paper delves into the game of Ms. Pac-Man. Ms. Pac-Man is a 1-person perfect-information non-deterministic video game. The goal of this paper is to develop a successful agent which is controlling Ms. Pac-Man. The paper proposes to apply Monte-Carlo Evaluations (MCE). In the experiments the parameters of the algorithm are tuned. The agent turned out to be weaker than the best knowledge-rule-based agent, ICE Pambush 3, on average.

Keywords: Ms. Pac-Man, Monte-Carlo Evaluations

1 Introduction

Ms. Pac-Man is a classic arcade video game that was released in 1980. The game has been used for competitions on various conferences, namely IEEE Conference on Computational Intelligence and Games (CIG) 2010, IEEE Conference on Computational Intelligence and Games (CIG) 2009, IEEE Congress on Evolutionary Computation (CEC) 2009, IEEE World Congress on Computational Intelligence (WCCI) 2008 and IEEE Congress on Evolutionary Computation (CEC) 2007 [10]. The contestants have to develop a software agent for Ms. Pac-Man which is tested on an identical computer with the goal to reach the highest score. First every program will be run 10 times. The 5 agents with the highest scores among those 10 runs will take part in the final. Each of the 5 agents reaching the final will be run an additional 3 times. The agent with the highest peak score over the total of 13 runs wins the contest. The best scoring agent to date was developed by Matsumoto, Ashida, Ozasa, Maruyama and Thawonmas of the Intelligent Computer Entertainment Laboratory, Department of Human and Computer Intelligent, Ritsumeikan University in Kyoto, Japan [11]. Their program, ICE Pambush 3, reached a high score of 30,010 at the CIG 2009. They claim to have an average score of 16,000 over the last 50 games played before the CIG 2009 contest. The agent used a rule-based approach combined with two versions of the

A* algorithm to calculate a path. The majority of these knowledge rules are focused on ambushing the ghosts by luring them to a power pill and then eating them. The goal of this paper is to develop an agent for Ms. Pac-Man with the objective to reach a high score. The paper proposes to use the Monte-Carlo Evaluations technique to create a high scoring agent. Monte-Carlo Evaluations have been used to much success in various board and card games, namely Othello [1], Tic-Tac-Toe [1], Chess [1], Go [5], Backgammon [16], Bridge [7, 15], Poker [3] and Scrabble [14]. Furthermore developing a strong evaluation function for Ms. Pac-Man is a difficult task. Thus the main research question is: *Can Monte-Carlo Evaluations successfully be applied to Ms. Pac-Man?*

This paper starts with a brief description of the game Ms. Pac-Man and all its components in Section 2. In Section 3 the game environment is discussed along with its competitive uses. Followed by the description of Monte-Carlo Evaluations (MCE) and how it is applied to Ms. Pac-Man in Section 4 and 5. In Section 6 the experiments are demonstrated and their results shown. Next, the conclusions are given in Section 7. Finally, in Section 8 a brief overview of possible future research is given.

2 A Brief Description of the Game Ms. Pac-Man

The original Pac-Man game was released in May 1980 in Japan [12], under the name Puck-Man. It was developed by Namco Ltd. as an arcade game. Midway Manufacturing Co. adopted it in the U.S. under the name Pac-Man in October 1980. Midway Manufacturing Co. later released an unofficial sequel in 1981, called Ms. Pac-Man (shown in Figure 1). The franchise was a big success worldwide and is known to be one of the biggest classics in the video gaming industry of all times. Only Ms. Pac-Man will be discussed in this paper. The game takes place in a predefined maze filled with pills, power pills and some special objects like fruits. The player controls a movable object in the maze named Pac-Man

(or Ms. Pac-Man). Its job is to eat as many pills as possible before it gets caught by a ghost. The more pills it eats the higher the score. Most commonly Ms. Pac-Man starts with three lives and the game ends when Ms. Pac-Man has zero lives left, meaning it gets caught three times (assuming no bonus lives are acquired).



Figure 1: The game at its initial state for the first maze.

2.1 The Ghosts

There are four ghosts in the maze trying to catch Ms. Pac-Man. All four are programmed using a different function to minimize the distance to player's location [10, 12, 4]. The first ghost, nicknamed Blinky and colored red, always tries to minimize the shortest path to Ms. Pac-Man. This is a simple and straightforward function, namely a standard Greedy algorithm. He is the chaser of the pack. The second ghost, nicknamed Pinky and thus colored pink, is programmed according to the Manhattan distance. The Manhattan distance is the absolute distance between two coordinates. The third ghost, nicknamed Inky and colored cyan, is programmed according to the Euclidean distance. The Euclidean distance is the square root of the squared difference in value of the x and the y coordinate. The fourth and last ghost, nicknamed Clyde and colored orange, is the ghost that exhibits a completely random behavior. This ghost is often referred to as being stupid.

2.2 Scoring

As with most games scoring is a vital part in Ms. Pac-Man. Ms. Pac-Man gains points by eating pills, which are spread throughout the maze. There is a fixed set of pills, which does not reappear once eaten. The level is complete once all pills are eaten. There are also four special pills in each level, called power pills. These pills grant Ms. Pac-Man the ability to eat the ghosts. Ms. Pac-Man only gains this ability for a short period of time, during which the ghosts turn blue and reverse their movement. Ms. Pac-Man gets bonus points for eating a ghost during this powered up period. Once a ghost gets eaten it moves to the center of the maze where it has to stay for a certain time period. Both the duration of the powered up phase and the ghosts having to stay in center after being eaten depend on the level. The higher the level the shorter those durations are. Finally, there are sometimes special objects in the center of the maze in the shape of fruits. These fruits give the player bonus points after being collected. The fruits are excluded for the experiments in this paper. While there are only four different mazes the level count is practically infinite. Every time the player completes the sequence of four mazes the game continues again at the first maze, but now the ghost's speeds are increased and the powered up duration and duration of ghosts staying in the center after being eaten are reduced.

3 The Game Environment

For this paper an already developed software toolkit, called Pac-Man, was used to model the game environment. This software toolkit was written by Lucas *et al.* [10]. This software toolkit is coded in the programming language Java. The toolkit consists of two modules, (1) *pacman* and (2) *screenpac*. The first and oldest module dates back to 2004. This module is now almost redundant for competitive use. A screenshot of the simulator of the *pacman* module is shown in Figure 2. The second and newer module is the one that is used for the Ms. Pac-Man competitions. The first module was used for this paper. The game code needed some rewriting and cleaning up in order to be properly used for this paper, i.e. removing unneeded code and print-outs. The game code incorporates all the objects described in the previous section with the exception of fruits. The software toolkit models the game maze using a node grid. Each node has its own index, (x,y) -coordinate and pill value. The pill value indicates whether that node is a pill, power pill or no pill at all. Ms. Pac-Man and the ghosts move from node to node, one step at a time. The game plays through the use of game cycles (in the first module). These cycles are not

bound by a fixed time interval. They simply represent the movement choice of the Ms. Pac-Man at that game state and then update the score and ghost positions accordingly. Game cycles are comparable with turns of board and card games. Due to the speed of the choice making process the game plays as if everything runs in real-time. The game ends when a predefined number of games cycles is reached or Ms. Pac-Man has zero lives left after which the score is returned in the console.

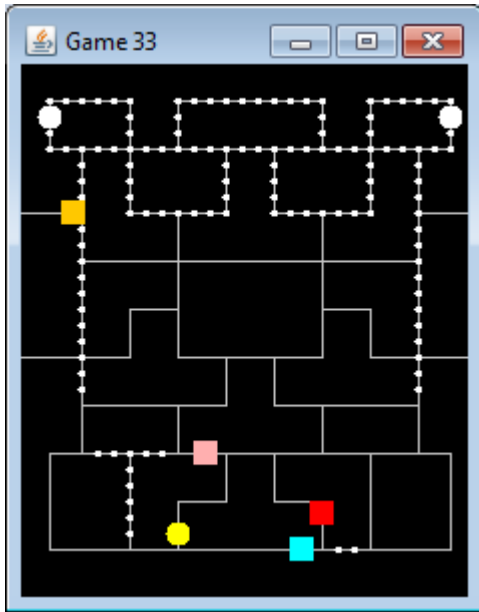


Figure 2: The game as it is shown in the simulator.

3.1 Competitions

Currently there are two competitions running based on Ms. Pac-Man, (1) the Ms. Pac-Man Screen-Capture Competition and (2) the Ms. Pac-Man versus Ghost Team Competition. Both competitions are programmed using the *screenpac* module.

The first competition has the longest runtime and has been held at: IEEE Conference on Computational Intelligence and Games (CIG) 2010, IEEE Conference on Computational Intelligence and Games (CIG) 2009, IEEE Congress on Evolutionary Computation (CEC) 2009, IEEE World Congress on Computational Intelligence (WCCI) 2008 and IEEE Congress on Evolutionary Computation (CEC) 2007. This competition is based around the idea of capturing the current game state through the pixels on the game window and then sending keyboard input to move Ms. Pac-Man. The screen is captured in a pixel map and then component analysis is applied in order to extract the positions of Ms. Pac-Man, ghosts, pills and power

pill. An additional processing is applied to determine in which directions Ms. Pac-Man is able to move in the current game state. Due to the scheduling of the screen captures it is possible that Ms. Pac-Man will not be able to move in the desired direction the program sends it, because Ms. Pac-Man might have moved in the time between the screen capture and the moment it receives its new desired direction from the program. Thus the programmer has to deal with another form of non-determinism other than that already inherent in the game of Ms. Pac-Man.

The second running competition is a rather new one which has not yet been run at a conference. Though it is running for IEEE Congress on Evolutionary Computation (CEC) 2011. This competition is only simulation based. The programmers have to design either (or both) an agent for Ms. Pac-Man or the Ghost Team. This makes the competition also an interesting test domain for multi-agent strategies. The goal of the Ghost Team agent is to minimize the score of Ms. Pac-Man, while Ms. Pac-Man's agent tries to maximize Ms. Pac-Man's score. The game runs through a simulator which sends the game state to both agents each game cycle. Currently there are 25 game cycles per second. Each agent will run in a separate thread and can take as long as it wants to respond. This naturally has the downside that the game state may have changed from the game state to which the agent calculated its desired move to by the time the calculation is done. The simulator waits 40 milliseconds every game cycle before it updates the game state with the new inputs given from the agents. This paper applies to this second competition and only designs a Ms. Pac-Man agent. It uses the *legacy* Ghost Team agent, which implies that the ghosts act as described in Subsection 2.1.

4 Monte-Carlo Evaluations

Monte-Carlo Evaluations (MCE) is an evaluation technique based on Monte-Carlo simulations. Monte-Carlo simulations are simulations that simulate a specified number of games in self-play using (quasi-)random game moves from the current game state onwards with the goal to determine the best next move. The best move typically is the move with the highest average score on the number of simulated games. MCE has been applied to various classic board and card games in the past, namely Othello [1], Tic-Tac-Toe [1], Chess [1], Go [5], Backgammon [16], Bridge [7, 15], Poker [3] and Scrabble [14]. Yet MCE has not often been applied to video games. MCE has a few drawbacks. The biggest drawback is that it heavily depends on the number of possible moves at every game state to be

satisfactory fast, because it takes more simulations in the case of many possible moves to keep the variance at an acceptable level to be able to choose the best next move. Two good ways to reduce this drawback are using (1) domain knowledge and (2) cutting off the simulated games before they reach the terminal state. The first technique simply decreases the randomness of the moves during the simulated games by improving the choice-making process so that you increase the quality of the simulation and thus get a more stable outcome. One way of doing this is never to consider a direction where Ms. Pac-Man runs into a ghost. The second technique basically allows you to run more simulations in the same time frame. The reason for this is that the simulated games take less computational time, since they come to a premature end. The easiest way to achieve this is by allowing the simulations to run for only a limited number of seconds or game cycles. By being able to run more simulations you also reduce the variance and thus get a more stable outcome.

5 The Application of MCE on Ms. Pac-Man

The application of Monte-Carlo Evaluations on the Ms. Pac-Man agent is rather straightforward. When moving through the maze and reaching a junction the agent runs a specified number of Monte-Carlo simulations. In addition simulations are also run when reaching a corner for the purpose of re-evaluation. During these simulations Ms. Pac-Man moves through the maze and chooses a random direction whenever it reaches a junction, though it is not allowed to choose the direction it came from. The agent runs a specified number of simulations and uses the *Selection Strategy* described in Subsection 5.1 to decide in which direction to simulate when iterating through the number of simulations. Based on the results of these simulations a desired direction is chosen. In this case the formula shown in Subsection 5.1 is also used for the final move selection. The agent is punished during the simulations if it chooses a direction where Ms. Pac-Man is eaten by subtracting a fixed value from the initial score (when the simulation started) and using that score as the final score for that given simulation. This boils down to having a worse score when the simulation ends than when the simulation started. Resulting in a decrease of the randomness of the simulations, because those directions are made less appealing. This also increases the overall quality of the simulations, as shown in the experiments in Subsection 6.2. Furthermore the simulations only run for a limited number of game cycles. This number depends on how much the current

level is completed, i.e. how much pills are already eaten. The more pills are eaten the higher the number of game cycles are made available for the simulations. The number of cycles runs from the lower bound 40 to the upper bound 240.

5.1 Selection Strategy

When deciding in which direction to simulate next the agent uses the following formula:

$$\operatorname{argmax}_i \bar{X}_i + \sqrt{\frac{\sum x_i^2 - t(N_i) \cdot \bar{X}_i^2 + D}{t(N_i)}}$$

This formula is a modified version of the UCT (Upper Confidence bound applied to Trees) formula used in Single-Player Monte-Carlo Tree Search by Schadd *et al.* [13]. Where i represents the direction. \bar{X}_i is the mean of the scores of the simulations for direction i . Sum x_i^2 is the squared sum of all the individual scores of the simulations for direction i . $t(N_i)$ is number of simulations performed for direction i . D is a high constant, for instance 1000.

6 The Experiments and Results

All the experiments were performed on the same computer, an Intel Core i7 920 processor (4x 2,66 GHz, C0 Stepping, 4x 256 kB L2 Cache, 8 MB shared L3 Cache, Intel HyperThreading, Intel Turbo Boost) coupled with 6 GB of 1333 MHz DDR3 RAM. Several tests have been run altering various parameters used by the agent. In all experiments testing is performed by playing 1000 games for each data point. In every game Ms. Pac-Man starts with 3 lives and does not have the ability to gain any additional lives on top of that.

In Subsection 6.1 the number of simulations is experimentally evaluated. Next, in Subsection 6.2 the *punishment value* is experimentally evaluated. Finally, in Subsection 6.3 the *D-value* is experimentally evaluated.

6.1 The Number of Simulations

In this subsection testing is performed to determine how much the final score depends on the number of simulations done. The experiments are conducted using a *punishment value* of 100 and a *D-value* of 1000. The lower bound for the number of game cycles per simulation is 40 and the upper bound is 240. In Table 1 Sims stands for the number of simulations and STD for

Test	Sims	Avg. Score	STD	High Score
1	10	4969.65	2303.09	14,180
2	20	5852.45	3159.12	24,440
3	30	6051.26	3364.52	27,620
4	40	6149.14	3279.81	22,020
5	50	6079.57	3211.19	23,020
6	60	6187.24	3257.51	20,770
7	70	6313.24	3368.22	24,450
8	80	6111.01	3378.11	22,350
9	90	6248.75	3309.27	20,000
10	100	6262.85	3206.87	21,650
11	110	6097.81	3371.32	23,840
12	120	6218.88	3427.21	21,300

Table 1: The results of testing different numbers of simulations.

the standard deviation.

Due to the (quasi-)random nature of the Monte-Carlo simulations the average score still fluctuates in the [6000-6400] range once you use 30 or more simulations instead of converging to a constant number. The standard deviation stabilizes towards the 3300 mark. For the remaining number of experiments the number of simulations used is 30 (highest peak score) and/or 70 (highest average score).

6.2 The Punishment Value

In this subsection testing is performed for the *punishment value*. The *punishment value* is the constant that is subtracted from the initial game score from which the simulation started should Ms. Pac-Man get eaten. That resulting score is then used as the final score for that given simulation. In the first series of experiments the number of simulations used is 30 and in the second series the number of simulations used is 70. The *D-value* is in both series equal to 1000. The lower bound for the number of game cycles per simulation is 40 and the upper bound is 240. In the Tables 2 and 3 Punish stands for the *punishment value* and STD for the standard deviation.

First of all, when not punishing the agent it cannot distinguish in the simulations between not eating any pills and surviving on the one hand and not eating any pills and being eaten on the other hand. This leads to overall bad play as shown by the results. Secondly, when the *punishment value* increases beyond 100 the average score starts to decrease. This can be explained by the fact that the agent is more actively minimizing the risks to the point where one punished simulation means a direction is not considered for further simulations

Test	Punish	Avg. Score	STD	High Score
1	0	3541.44	1185.57	10,180
2	100	6340.41	3311.31	20,210
3	200	6031.03	3270.48	22,370
4	300	6115.08	3234.00	24,940
5	400	5768.44	3071.07	21,590
6	500	6017.68	3063.72	20,800
7	1,000	5760.18	3034.26	21,610
8	2,000	5635.94	2859.40	24,140
9	5,000	5387.45	2766.55	16,650
10	10,000	5567.42	2888.79	17,790

Table 2: The results of testing different punishment values when using 30 simulations.

Test	Punish	Avg. Score	STD	High Score
1	0	3343.25	1070.67	8,980
2	100	6304.64	3235.94	22,030
3	200	6050.12	3169.75	26,070
4	300	5944.86	3163.81	30,110
5	400	5865.38	3011.96	20,150
6	500	5608.97	2849.28	19,900
7	1,000	5626.22	2849.78	20,060
8	2,000	5512.67	2847.21	24,320
9	5,000	5441.74	2854.20	22,860
10	10,000	5341.15	2680.55	22,360

Table 3: The results of testing different punishments value when using 70 simulations.

anymore. While that direction might not be a bad direction at all.

In the second series, the highest score of all games run using the MCE agent was obtained, namely 30,110. This high score is higher than the high score obtained by ICE Pambush 3 at the CIG 2009 of 30,010. Regardless of this, the average score is still notably lower than the acclaimed average score of ICE Pambush 3, which is 16,000.

6.3 The D Value

In this subsection testing is performed on the *D-value*. The number of simulations used in all tests is 70 and the *punishment value* is equal to 100. The lower bound for the number of game cycles per simulation is 40 and the upper bound is 240. In Table 4 STD stands for standard deviation.

The results show that the *D-value* is required for the successfulness of the algorithm. However, the exact value of the *D-value* is less important, it only needs to be within the [100-10,000] range. For instance, in Test 3 the average score is the highest, but at the same time

Test	D-value	Avg. Score	STD	High Score
1	0	3709.44	1334.85	9,950
2	100	6220.21	3213.98	20,090
3	500	6256.90	3226.11	23,200
4	1,000	6098.19	3077.00	20,300
5	2,000	6094.65	3014.93	18,860
6	3,000	6375.49	3480.80	21,860
7	4,000	5983.81	3092.00	19,260
8	5,000	5995.26	3152.78	21,840
9	10,000	6001.32	3172.97	24,040
10	50,000	5373.29	2679.20	17,780
11	10,0000	3946.44	1383.80	10,120

Table 4: The results of testing different D-values.

the standard deviation is also the highest, thus that result can be allocated to the (quasi-)random nature of the simulations. Once the *D-value* exceeds 10,000 the average game score starts to drop significantly.

The highest average score of all 43,000 test runs was reached using a *D-value* of 3,000, but at the same time the standard deviation is also the highest of all the test runs. So this result can also be allocated to the (quasi-)random nature of the Monte-Carlo simulations.

7 Conclusion

The research question of this paper was: *Can Monte-Carlo Evaluations successfully be applied to Ms. Pac-Man?* As shown in this paper Monte-Carlo Evaluations can be applied to the 1-person perfect-information non-deterministic video game Ms. Pac-Man. The agent developed even beat the high score of the ICE Pambush 3 agent of 30,010, obtained at the CIG 2009, once out of the 43,000 test runs. However on average it still performs worse than the best knowledge-rule-based agent, with the best average score of 6375.49 versus the acclaimed 16,000 of ICE Pambush 3.

Testing has been performed on the various parameters of the MCE algorithm. The experiments show that for the number of simulations the optimal value is 70, though due to the (quasi-)random nature of the simulations any number above 30 might give good results. Mentioned must be that if the number chosen is too high the game will be slowed down notably. Testing the *punishment value* has shown that 100 is the best choice. It perfectly lies on the balance between taking risks, but not punishing the agent excessively when it makes a mistake, and not playing too conservative and cautious. Finally, for the *D-value* experiments showed that it has to be below 10,000 to get decent results, preferably within the [100-3000] range.

To summarize the results: Monte-Carlo Evaluations is a

decent way of overcoming the difficulty of developing an evaluation function for Ms. Pac-Man. The highest score reached during the experiments was 30,110. Human experts however are no match for the agent, reaching scores of several 100,000's.

8 Future Research

One of the goals of future research would be extending the current agent with the use of Monte-Carlo Tree Search (MCTS). Monte-Carlo Tree Search is a relatively new technique used to overcome the difficulty of developing a sufficiently fast heuristic evaluation function for non-terminal game states in various board and video games. Based on the results of the Monte-Carlo Evaluations, MCTS could be a promising direction to improve the agent further. MCTS has been applied to Go [6], Amazons [8], Lines of Action [17], Havannah [9], Hex [2] and other games.

References

- [1] Abramson, B. (1990). Expected-Outcome: A General Model of Static Evaluation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 12, No. 2, pp. 182–193.
- [2] Arneson, B., Hayward, R.B. and Henderson, P. (2010). Monte Carlo Tree Search in Hex. *IEEE Transactions on Computational Intelligence and AI in Games*, Vol. 2, No. 4, pp. 251–258.
- [3] Billings, D., Peña, L., Schaeffer, J. and Szafron, D. (1999). Using Probabilistic Knowledge and Simulation to Play Poker. *Proceedings of the Sixteenth National Conference on Artificial Intelligence and the Eleventh Innovative Applications of Artificial Intelligence Conference Innovative Applications of Artificial Intelligence*, AAAI '99/IAAI '99, pp. 697–703, American Association for Artificial Intelligence, Menlo Park, CA, USA.
- [4] Birch, C. (2010). Understanding Pac-Man Ghost Behavior. <http://gameinternals.com/post/2072558330/understanding-pac-man-ghost-behavior>.
- [5] Brüggemann, B. (1993). Monte Carlo Go. Technical report, Physics Department, Syracuse University, Syracuse, NY, USA.
- [6] Chaslot, G.M.J-B., Winands, M.H.M., Herik, J.H. 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.

- [7] Ginsberg, M.L. (1999). GIB: Steps Toward an Expert-Level Bridge-Playing Program. *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, IJCAI '99*, pp. 584–593, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.
- [8] Lorentz, R. (2008). Amazons Discover Monte-Carlo. *Computers and Games (CG 2008)*, Vol. 5131 of *Lecture Notes in Computer Science*, pp. 13–24. Springer Berlin, Heidelberg.
- [9] Lorentz, R. (2011). Improving Monte-Carlo Tree Search in Havannah. *Computers and Games (CG 2010)*, Vol. 6515 of *Lecture Notes in Computer Science*, pp. 105–115. Springer Berlin, Heidelberg.
- [10] Lucas, S.M. (2010). Ms Pac-Man versus Ghost-Team Competition. <http://cswww.essex.ac.uk/staff/sml/pacman/kit/AgentVersusGhosts.html>. School of Computer Science and Electronic Engineering, University of Essex, Colchester, Essex, UK.
- [11] Matsumoto, H., Ashida, T., Ozasa, Y., Maruyama, T. and Thawonmas, R. (2009). ICE Pambush 3. Technical report, Intelligent Computer Entertainment Laboratory, Department of Human and Computer Intelligent, Ritsumeikan University, Kyoto, Japan.
- [12] Pittman, J. (2011). The Pac-Man Dossier. <http://home.comcast.net/~jpittman2/pacman/pacmandossier.html>.
- [13] Schadd, M.P.D., Winands, M.H.M., Herik, J.H. van den, Chaslot, G.M.J-B. and Uiterwijk, J.W.H.M. (2008). Single-Player Monte-Carlo Tree Search. *Proceedings of the 6th International Conference on Computers and Games, CG '08*, pp. 1–12, Springer-Verlag, Berlin, Heidelberg.
- [14] Sheppard, B. (2002). *Towards Perfect Play of Scrabble*. Ph.D. thesis, Maastricht University, Maastricht, The Netherlands.
- [15] Smith, S.J.J., Nau, D. and Throop, T.A. (1998). Computer Bridge: A Big Win for AI Planning. *AI Magazine*, Vol. 19, No. 2, pp. 93–106.
- [16] Tesauro, G. and Galperin, G. (1996). On-Line Policy Improvement Using Monte-Carlo Search. *Advances in Neural Information Processing Systems 9*, pp. 1068–1074, MIT Press, Cambridge, MA, USA.
- [17] Winands, M.H.M. and Björnsson, Y. (2010). Evaluation Function Based Monte-Carlo LOA. *Advances in Computer Games (ACG 2009)*, Vol. 6048 of *Lecture Notes in Computer Science*, pp. 33–44. Springer Berlin, Heidelberg.