Helping AI to Play Hearthstone: AAIA'17 Data Mining Challenge

Andrzej Janusz*†, Tomasz Tajmajer‡†
*Institute of Informatics, University of Warsaw
Banacha 2, 02-097 Warsaw, Poland
{janusza,t.tajmajer}@mimuw.edu.pl

Maciej Świechowski†
†Silver Bullet Solutions
Liwiecka 25, 04-289 Warsaw, Poland
m.swiechowski@mini.pw.edu.pl

***Abstract*—This paper summarizes the AAIA'17 Data Mining Challenge: Helping AI to Play Hearthstone which was held between March 23, and May 15, 2017 at the Knowledge Pit platform. We briefly describe the scope and background of this competition in the context of a more general project related to the development of an AI engine for video games, called Grail. We also discuss the outcomes of this challenge and demonstrate how predictive models for the assessment of player's winning chances can be utilized in a construction of a smart bot for playing Hearthstone. Finally, we show results of our recent experiments whose objective was to verify whether the models trained on data generated by bots used in our competition, can be used to make predictions in games played by different (i.e. more advanced) types of bots.**

***Keywords*—data mining competition; AI in video games; MCTS; artificial neural networks; Hearthstone: Heroes of Warcraft;**

I. Introduction

HEARTHSTONE: HEROES OF WARCRAFT is a free-to-play online video game developed and published by Blizzard Entertainment. It is an example of a turn-based collectible card game played between two opponents. During a game, players use their custom decks of thirty cards, along with a selected hero with a unique power. They spend mana points to cast spells or summon minions to attack the opponent, with the goal to reduce the opponent's health to zero. Building efficient decks is an essential skill and many archetypes of decks exists. These archetypes are characterized by different distributions of the card's mana cost and thus are meant for players with different play styles. There are also sets of cards, which synergize well due to their unique properties and can be used in many different decks.

In recent years, Hearthstone has become a testbed for AI research. A community of passionate players and developers have started the HearthSim project (https://hearthsim.info/) and created many tools that allow simulating the game for the purpose of AI and machine learning experiments. Several researchers have already used this game in their studies [1], [2]. Moreover, our research team decided to use Hearthstone as one of case studies which aim to demonstrate capabilities of our video game's AI designing framework, called Grail. For this reason, one objective of this article is to explain how some powerful heuristic search algorithms can be combined with prediction models that derive from the machine learning domain, in order to construct a smart and cunning artificial Hearthstone player.

In the following sections...

II. AAIA'17 Data Mining Challenge

AAIA'17 Data Mining Challenge: Helping AI to Play Hearthstone (https://knowledgepit.fedcsis.org/contest/view.php?id=120) took place between March 23 and May 15, 2017. It was organized under auspices of 12^{th} International Symposium on Advances in Artificial Intelligence and Applications (AAIA'17, https://fedcsis.org/2017/aaia) which is a part of the FedCSIS conference series.

The main objective in this competition was to construct a prediction model which would be able to foresee who is going to win, using only information about a single game state. The ability to accurately assess winning chances of a player in different game states is substantial for designing efficient and challenging AI opponents in many video games. The most famous example is the AlphaGo program, which used two neural networks to evaluate possible moves and game states of Go games [3]. In our competition, we challenged participants with the task to design such models for Hearthstone.

In particular, the data set provided to participants contained examples of game states extracted from Hearthstone play outs between weak AI players (i.e. the bots which were used to generate the data were choosing their in-game decisions at random). The participants were asked to predict winning chances of the first player from game states belonging to the test set and submit their predictions to the Knowledge Pit competition platform [4]. In order to give participants a freedom of choosing a representation of the data which they want to use, apart from a preprocessed data sets in a tabular format, we provided raw JSON files which described particular game states in more details.

The training part of the data was made available along with the corresponding information regarding the actual game winners. These labels were removed from the test set which was also made available to participants. Initially, the training set consisted of 2000000 game states, however, after detecting an unwitting data leakage [5], after first few weeks of the challenge, it was extended by additional 1250000 cases from the original test set (in total, there were 3250000 training examples). The final test set consisted of 750000 game states. These examples were obtained from a different set of Hearthstone play outs than the training cases. In fact, while the training data contained game states from ≈ 65000 simulations, more than 180000 play outs were simulated to generate the test set. It is also important to note that while in the training games there were used only 9 different sets of cards (one deck for

TABLE I. BASIC CHARACTERISTICS OF DATA SETS USED IN AAIA'17 DATA MINING CHALLENGE.

characteristic	training set	test set
no. examples	3250000	750000
no. games	65000	180000
no. used decks	9	27
percent of wins	50.46%	57.27%
min. win rate per hero (percent/id)	50.07%/326	37.53%/754
max. win rate per hero (percent/id)	50.85%/981	75.34%/25

TABLE II. FINAL RESULTS AND NUMBER OF SUBMISSIONS FROM THE TOP RANKED TEAMS. THE LAST ROW SHOWS THE RESULT OBTAINED BY OUR BASELINE SOLUTION – A FULLY-CONNECTED NEURAL NETWORK WITH TWO HIDDEN LAYERS, TRAINED ON THE TABULAR DATA.

team name	rank	# of submissions	final result
iwannabetheverybest	1	139	0.8019
hieuvq	2	384	0.7992
johnpateha	3	143	0.7990
vz	4	11	0.7973
jj	5	75	0.7971
karek	6	16	0.7968
podludek	7	19	0.7966
...	...	...	...
baseline	94	–	0.7846

every hero type), the test games were played using 27 different decks. As a consequence, the test data contained Hearthstone cards which had never appeared in the training set. Table I shows a summary of basic characteristics of data sets used in the challenge.

A. Evaluation of results and participation in the challenge

Participants of the competition had to prepare their solutions in a form of a file with predictions of a likelihood that *player 1* will win, given a corresponding description of a game state. The files with predictions had to be sent using the submission system of Knowledge Pit [4]. Each of the competing teams could submit multiple solutions. Quality of the submissions was measured using Area Under the ROC Curve (AUC) [6]. The submitted solutions were evaluated on-line and the preliminary results were published on the competition Leaderboard. The preliminary score was computed on a subset of the test set, fixed for all participants. Size of this subset corresponded to randomly chosen 5% of the test set. The final evaluation was conducted after completion of the competition using the remaining part of the test data.

Apart from submitting their predictions, each team was also obligated by competition rules to provide a brief report describing its approach. Only the final solutions from teams which sent a valid report could undergo the final evaluation and be published among the competition results. In this way, we were able to collect a vast amount of information regarding efficient representation methods of Hearthstone game states and state-of-the-art approaches to this type of prediction problems.

B. Summary of the competition

Even though AAIA'17 Data Mining Challenge lasted for less than two months, it attracted attention of many researchers from domains of machine learning and artificial intelligence in video games. By the end of competition there were 296 teams from 28 countries registered in the challenge. Among them, 188 teams submitted at least one solution to the public leaderboard and 114 teams described their solution in a report uploaded to the Knowledge Pit platform. In total, we received 4067 submission, which makes this competition the most popular one among challenges organized at Knowledge Pit to this day.

The large number of submitted reports gave us a unique opportunity to review the most effective prediction methods for the assessment of Hearthstone game states. The most successful approach in this regard turned out to be artificial neural networks [7] and particularly, the deep learning methods [8]. In fact, all top-ranked teams used neural networks in their solutions and the winners focused only on the convolutional neural networks [9]. Another popular approach was the utilization of *xgboost* algorithm [10]. There were also much simpler approaches which turned to be efficient, such as the logistic regression models. Moreover, All of these to methods were often combined – techniques such as averaging, bagging or stacking were commonly used to obtain better prediction results [11]. Table II presents scores obtained by the seven top-ranked teams. Noticeable is the fact that the difference in scores between the best solution and the baseline is less than 2%.

Many teams decided to use data in the JSON format in order to construct richer representations of game states than the one which was available in the provided tabular data. Extracted features were often a reflection of participant's experience and domain knowledge about Hearthstone. Their descriptions included in reports turned out to be a valuable source of knowledge which can be used to improve our artificial Hearthstone players.

III. AUGMENTATION OF GAME STATE SEARCH HEURISTICS WITH NEURAL NETWORKS

A. Monte-Carlo Tree Search

Monte Carlo Tree Search (MCTS) [12] is a method of learning an optimal policy in a decision problem such as a game-playing problem. It combines tree search, random sampling and statistical analysis. For the first time, it was used for games in Go [13] as an improvement over a Monte Carlo sampling technique (without the tree search) [14]. The algorithm has led to a breakthrough in the game of Go, which had been previously regarded as practically impossible for computer programs to play effectively [15]. Driven by this success, MCTS has become the state-of-the-art approach in various game domains [16] such as Go [13], [17], Arimaa [18], Havannah [19], as well as General Game Playing [20], [21] and General Video Game Playing [22].

The idea of MCTS is to repeatedly simulate the game (problem) and build statistics about states and actions. Each iteration of the algorithm consists of four phases as depicted in Figure 1.

1) **Selection.** In this phase, the algorithm starts from the root node and searches the tree down by choosing

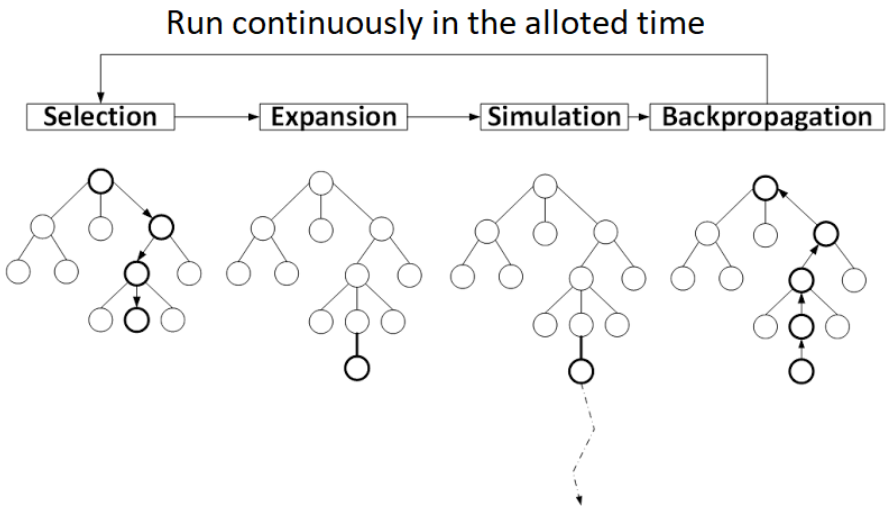

Fig. 1. Depiction of four phases which comprise the MCTS algorithm.

subsequent children nodes. The child node at each node down the path is chosen according to the so-called selection policy. The selection phase ends when there is no child node to choose, i.e., a leaf node has been reached.

2) **Expansion.** One of the possible actions is applied to a node selected in the previous step and the tree is grown by adding a child node representing the resulting state.
3) **Simulation.** The algorithm starts from the new node and performs a complete game simulation, i.e., reaching a terminal state. This phase is done outside the game-tree and no nodes are added to it. Once the simulation reaches the terminal state, the obtained goals (outcomes) of each player are fetched.
4) **Back-propagation.** Here, the statistics are recalculated inside all nodes along the path from the root to the leaf (containing the starting state for the simulation) in the game tree. The statistics include the average scores of each player and the number of visits to a node. An average score is computed as the total score achieved in iterations going through a particular node divided by the number of visits to that node.

In the classical implementation, actions in the simulation phase are chosen with respect to uniform random distribution. In the selection phase, a more sophisticated formula (selection policy) is typically used. The most common one, which was also employed in this paper for all MCTS-based programs used during the experiments is called Upper Confidence Bounds applied for Trees (UCT) [23], [24].

$$a^* = \arg\max_{a \in A(s)} \left\{ Q(s,a) + C\sqrt{\frac{ln\,[N(s)]}{N(s,a)}} \right\} \qquad (1)$$

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.

The MCTS algorithm using the UCT selection formula is proved to converge to the min-max theoretical optimum [23]. However, it poses several advantages over a classical min-max search. For instance, it does not require any game specific evaluation function and constructs the tree in an asymmetric manner, focusing at the most promising lines of play. It scales better with the depth of the tree, it can be stopped at anytime to return the best action found so far.

B. Monte-Carlo Tree Search with Heuristics

Despite the wide usage in a variety of game domains, the MCTS method has bottlenecks and limitations. It is both computationally demanding and memory intensive. Games with huge branching factor, i.e., the total number of actions available to players, in average, often inhibit the usage of MCTS and other tree-search methods. 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 are 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:

(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) Move Ordering - the heuristic defines the order, in which actions in the tree are expanded (chosen for the first time). This method has the most significant impact on the deeper parts of the tree, because the MCTS is less likely to visit them again, so the order matters. If better moves are expanded first, their proximity in the tree has a higher chance to be visited in subsequent simulations.

(3) Simulation Policy Bias - in the baseline version of the MCTS algorithm, the actions during the simulation phase are chosen randomly. With a good heuristic evaluation, a sensible approach is to infer this knowledge in the action selection process, while still leaving some degree of randomness. The two most common implementations are pseudo-roulette selection with probabilities computed using Boltzmann distribution (here the heuristic evaluation is used) or the so-called epsilon-greedy approach. 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 $1 - \epsilon$.

(4) Early Cutoff - the authors of [25] achieved the best results with terminating Monte Carlo simulations before the game ends and returning the heuristic evaluation of the current state. This variant is called Early Cutoff and the cutoff is done typically at fixed depth or with certain small probability (e.g. P=0.1) in each step.

The aforementioned AlphaGo program [3] employs both Tree Policy Bias and Simulation Policy Bias.

C. Generality of prediction models trained on random simulations

D. Learning playing strategy from sequences

Both Tree Policy Bias and Simulation Policy Bias methods utilize a heuristic function which provide the value of a state-action pair. It is however desired to have a function that provides the policy rather than the value of a particular action. The policy $\pi(a|s)$ specifies the probabilities for all actions available in a given state, thus it enables the selection of the best action candidate in a single state evaluation.

Various machine learning methods may be used to obtain the policy. Reinforcement Learning is used in particular for cases where the optimal policy is unknown and needs to be learned based on sparse reward signals. However, a supervised learning approach may be used, when examples of policies are available (e.g. from human players of other algorithms). In our case, we may use MCTS to generate Hearthstone matches and obtain state-action pairs i.e. record what action was chosen by MCTS as a response to given game state. Next, we may train a model that predicts the action that MCTS would choose for a given state. Training such a model is basically a classification task, well fitted for deep neural networks (DNN).

Long short-term memory (LSTM) is a type of recurrent neural network [27] dedicated for use with sequences. The architecture of LSTM enables it to learn long and short term temporal dependencies. LSTM provides superior performance in tasks such as speech recognition, machine translation or language modeling. A deep LSTM network may be created by stacking multiple LSTM layers.

A DNN may be trained to approximate a policy from examples of state-action pairs - we will refer to this network as to *policy network*. However, in case of Hearthstone, a single state may not provide enough information to the model for a valid action prediction. This is due to the fact that a single turn in HS consists of many moves. Moreover, a single move is decomposed into a few atomic actions in order to be efficiently implemented in the HS simulator. For example, putting a minion on the board consists of two actions: selecting a card from hand and selecting the slot on the board where the minion should be placed. To improve the accuracy of the policy predictions, rather than using a single state, we chosen to use a sequence of states and previous actions as the input to the policy network.

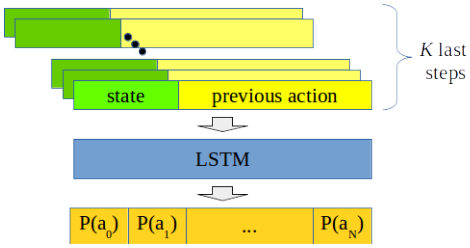

Fig. 2. LSTM policy network. A sequence of states and actions from previous game steps are provided as the input the LSTM network. The LSTM network is trained to output prediction of the action to be performed after the last step in the sequence.

Our policy network is presented in figure 2. LSTM network is provided with a sequence of $K = 10$ vectors created from concatenating a state vector for state s_{t-k} and action vector for action a_{t-k-1}, for $k \in [0, 1, ..., K-1]$. The state vector includes 403 values representing the state of the game from the point of view of a selected player. The action vector is of length 91 and contains a one-hot encoded action. The output of the LSTM is a single vector with probabilities assigned to each of all available 91 actions (the probabilities are provided also for actions that are illegal in a certain state). The LSTM in our experiments consisted of 3 layers of 256 LSTM cells with dropout

To obtain the training data, we first used MCTS with 1000 iterations to generate 7000 games between randomly selected decks. Using this data we obtained 528365 sequences of length 10 where each element of the sequence included 494 values. We will refer to this dataset as to *SeqMCTS1k*. Next, we generated 1500 games using MCTS with 10000 iterations. As a result we created a second dataset: *SeqMCTS10k*, that consists of 149521 sequences.

To evaluate the policy network, we created a greedy DNN agent that always selects the most promising action from the predictions of the policy network. We confronted this agent against a random agent and an agent using MCTS with 1000 iterations. Greedy DNN agents were using policy networks trained in three variants: 1) using only *SeqMCTS1k* dataset, 2) using only *seqMCTS10k* dataset and 3) trained first on the *SeqMCTS1k* dataset and then reatrained on the *SeqMCTS10k* dataset. The results of the evaluation are presented in Table III. Each score is calculated based on 500 games played between the agents.

TABLE III. EVALUATION RESULTS OF AGENTS USING LSTM POLICY NETWORK.

greedy agent type	wins vs random agent	wins vs MCTS(1000)
seqMCTS1k	96.2%	23.0%
seqMCTS10k	98.6%	26.4%
seqMCTS1k retrained with seqMCTS10k	98.2%	50.4%

IV. CONCLUSIONS

ACKNOWLEDGMENTS

This research was financially supported by the Polish National Centre for Research and Development (NCBiR) project POIR.01.02.00-00-0150/16 within the GameInn Program and by the Silver Bullet Solutions company.

REFERENCES

[1] D. Taralla, "Learning artificial intelligence in large-scale video games: A first case study with hearthstone: Heroes of warcraft," Ph.D. dissertation, Université de Liege, Liege, Belgium, 2015.

[2] 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.

[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] A. Janusz, D. Slezak, S. Stawicki, and M. Rosiak, "Knowledge pit - A data challenge platform," in *Proceedings of the 24th International Workshop on Concurrency, Specification and Programming, Rzeszow, Poland, September 28-30, 2015.*, 2015, pp. 191–195. [Online]. Available: http://ceur-ws.org/Vol-1492/Paper_18.pdf

[5] S. Kaufman, S. Rosset, C. Perlich, and O. Stitelman, "Leakage in data mining: Formulation, detection, and avoidance," *TKDD*, vol. 6, no. 4, p. 15, 2012. [Online]. Available: http://doi.acm.org/10.1145/2382577.2382579

[6] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, ser. Springer Series in Statistics. New York, NY, USA: Springer New York Inc., 2001.

[7] M. S. Szczuka and D. Slezak, "Feedforward neural networks for compound signals," *Theor. Comput. Sci.*, vol. 412, no. 42, pp. 5960–5973, 2011. [Online]. Available: https://doi.org/10.1016/j.tcs.2011.05.046

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems*, ser. NIPS'12. USA: Curran Associates Inc., 2012, pp. 1097–1105. [Online]. Available: http://dl.acm.org/citation.cfm?id=2999134.2999257

[9] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, "Large-scale video classification with convolutional neural networks," in *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition*, ser. CVPR '14. Washington, DC, USA: IEEE Computer Society, 2014, pp. 1725–1732. [Online]. Available: http://dx.doi.org/10.1109/CVPR.2014.223

[10] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16. New York, NY, USA: ACM, 2016, pp. 785–794. [Online]. Available: http://doi.acm.org/10.1145/2939672.2939785

[11] A. Janusz, "Combining multiple predictive models using genetic algorithms," *Intelligent Data Analysis*, vol. 16, no. 5, pp. 763–776, 2012. [Online]. Available: http://dx.doi.org/10.3233/IDA-2012-0550

[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] S. Gelly, Y. Wang, O. Teytaud, M. U. Patterns, and P. Tao, "Modification of UCT with Patterns in Monte-Carlo Go," 2006.

[14] B. Brügmann, "Monte Carlo Go," Citeseer, Tech. Rep., 1993.

[15] X. Cai and D. C. Wunsch II, "Computer Go: A Grand Challenge to AI," in *Challenges for Computational Intelligence*. Springer, 2007, pp. 443–465.

[16] N. R. Sturtevant, "An Analysis of UCT in Multi-Player Games," in *International Conference on Computers and Games*. Springer, 2008, pp. 37–49.

[17] S. Gelly *et al.*, "The Grand Challenge of Computer Go: Monte Carlo Tree Search and Extensions," *Commun. ACM*, vol. 55, no. 3, pp. 106–113, Mar. 2012, DOI: 10.1145/2093548.2093574.

[18] 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.

[19] F. Teytaud and O. Teytaud, "Creating an Upper-Confidence-Tree Program for Havannah," in *Advances in Computer Games*. Springer, 2009, pp. 65–74.

[20] 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.

[21] M. Świechowski, H. Park, J. Mańdziuk, and K.-J. Kim, "Recent Advances in General Game Playing," *The Scientific World Journal*, vol. 2015, 2015.

[22] D. Perez, S. Samothrakis, and S. Lucas, "Knowledge-Based Fast Evolutionary MCTS for General Video Game Playing," in *2014 IEEE Conference on Computational Intelligence and Games*. IEEE, 2014, pp. 1–8.

[23] L. Kocsis and C. Szepesvári, "Bandit Based Monte-Carlo Planning," in *Proceedings of the 17th European conference on Machine Learning*, ser. ECML'06. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 282–293.

[24] L. Kocsis, C. Szepesvári, and J. Willemson, "Improved Monte-Carlo Search," *Univ. Tartu, Estonia, Tech. Rep*, vol. 1, 2006.

[25] K. Walędzik 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, Sept 2014.

[26] G. M. J. Chaslot, M. H. Winands, H. J. V. D. HERIK, J. W. 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] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997. [Online]. Available: http://dx.doi.org/10.1162/neco.1997.9.8.1735