





# Predicting Opponent Moves for Improving Hearthstone AI

Alexander Dockhorn , Max Frick ,  
Ünal Akkaya , and Rudolf Kruse 

Otto-von-Guericke University,  
Institute for Intelligent Cooperating Systems,  
Universitaetsplatz 2, 39106 Magdeburg, Germany  
{alexander.dockhorn, max.frick, uenal.akkaya, rudolf.kruse}@ovgu.de

**Abstract.** Games pose many interesting questions for the development of artificial intelligence agents. Especially popular are methods that guide the decision-making process of an autonomous agent, which is tasked to play a certain game. In previous studies, the heuristic search method Monte Carlo Tree Search (MCTS) was successfully applied to a wide range of games. Results showed that this method can often reach playing capabilities on par with humans or even better. However, the characteristics of collectible card games such as the online game Hearthstone make it infeasible to apply MCTS directly. Uncertainty in the opponent’s hand cards, the card draw, and random card effects considerably restrict the simulation depth of MCTS. We show that knowledge gathered from a database of human replays help to overcome this problem by predicting multiple card distributions. Those predictions can be used to increase the simulation depth of MCTS. For this purpose, we calculate bigram-rates of frequently co-occurring cards to predict multiple sets of hand cards for our opponent. Those predictions can be used to create an ensemble of MCTS agents, which work under the assumption of differing card distributions and perform simulations according to their assigned distribution. The proposed ensemble approach outperforms other agents on the game Hearthstone, including various types of MCTS. Our case study shows that uncertainty can be handled effectively using predictions of sufficient accuracy, ultimately, improving the MCTS guided decision-making process. The resulting decision-making based on such an MCTS ensemble proved to be less prone to errors by uncertainty and opens up a new class of MCTS algorithms.

**Keywords:** Hearthstone, Monte Carlo Tree Search, knowledge-base, ensemble, uncertainty, bigrams

## 1 Introduction

Computational intelligence in games is a thriving research topic, with a growing demand from the video game industry. Especially, applications in video games



Fig. 1: Elements of the Hearthstone game board: (1) weapon slot (2) hero (bottom: player, top: opponent) (3) opponent's minions, (4) player's minions, (5) hero power, (6) hand cards, (7) mana, (8) decks, (9) history

make it possible to quickly compare agents, and their related methods, by letting them play against each other. The development of artificial intelligence (AI) agents for games often involves solving decision-making tasks, for which heuristic search processes, such as Monte Carlo Tree Search (MCTS) are frequently used [2].

Recent studies showed that the actual skill-level of an MCTS agent is dependent on the quality of performed simulations during the search [7]. This is especially hard in the context of collectible card games, such as the online game Hearthstone, where the set of playable card-combinations is extremely large. Even more critical is the high amount of uncertainty involved in the game. Game-mechanics such as the random card-draw, the unknown hand cards of our opponent, as well as random card effects hinder the accuracy of performed simulations and restrict the simulation depth.

In the context of a 2-player card game, the prediction of our opponent's moves, his hand-cards, as well as the cards in his deck largely influence which moves the player needs to consider. In this work we create a knowledge-base of frequently played card combinations from a database of human player game replays. During a game the knowledge-base is used to predict the opponent deck based on previously played cards. From the estimated deck we sample multiple hand card sets for our opponent. An ensemble of MCTS procedures is initialized based on the hand card samples. The ensemble's best result will be returned as final outcome of the decision-making process.

The developed agent is exemplary for uncertainty handling in MCTS agents. The result of the simulation is less prone to wrong assumptions, due to the included knowledge-base and the search on an ensemble of MCTS procedures. Our results indicate that the prediction-based ensemble improves the playing capabilities of our developed agent.

The remainder of this paper is structured as follows: In Section 2 we review the game Hearthstone: Heroes of Warcraft and previous research efforts on the development of Hearthstone AIs. Section 3 covers the Flat Monte Carlo and the Monte Carlo Tree Search algorithm. Additionally, we shortly review recently used agents found in the literature in Section 4. In Section 5 we first discuss our approach to learn frequent card combinations from a large database of replays. We further introduce our roll-out policy, which includes the prediction of our opponent’s cards based on our knowledge database. The influence of the new rollout policy and general parameters of MCTS is tested in Section 6 followed by a detailed discussion of our results. Finally, in Section 7 we highlight the implications of our work and summarize our suggestions for further improvements.

## 2 Hearthstone: Heroes of Warcraft

Hearthstone is a turn-based digital collectible card game developed and published by Blizzard Entertainment [1]. Players compete in one versus one duels choosing a self-constructed deck and one out of nine available heroes. In these matches players try to beat their opponents by reducing the opponent’s health from 30 to 0. This can be achieved by playing cards from the hand onto the game board at the cost of mana. Played cards can be used to inflict damage to the enemies hero or to destroy cards on his side of the game board. The amount of mana available to the player increases every turn (up to a maximum of 10), which also increases the complexity of later turns. At the beginning of a turn the player also draws a new card until his deck is empty. If no cards remain he receives *fatigue*-damage, which is steadily increasing from turn to turn. The standard game board is shown in Figure 1.

A deck is a self-constructed set of 30 cards, which can be chosen from more than 1000 cards currently included in the game. Each card brings unique effects to influence the current game board. Additionally, heroes can use class-specific hero-powers and cards.

Cards can be of the type minion, spell, or weapon. Figure 2 shows one example of each card type. Minion cards assist and fight on behalf of the hero. They usually have an attack-, health-, and mana cost-value, as well as additional abilities or a special minion type. Once played, they can attack the enemies side of the board every consecutive turn to inflict damage on either enemy minions or the opponent’s hero. Attacking also reduces its own health points by the attack points of the defender. In case a minion’s health drops to zero or less, it is removed from the board and put into the player’s graveyard. Spell cards can be cast using mana to activate various abilities and are discarded after use. They can have a wide range of effects, such as raising a minion’s attack, inflicting damage to a hero or minion, etc. Secrets, which are a special kind of spell, can be played without immediately activating their effect. Given a certain event a secret will be activated and for example destroy an attacking minion. Once activated, the secret is removed from the board. Weapon cards are directly attached to the player’s hero and also enable him to attack. Their durability value limits the



Fig. 2: General card types: cards include (1) mana cost , (2) attack damage, (3) health/durability, (4) and special effects.

number of attacks till the weapon breaks. Only one weapon can be equipped at the same time.

Hearthstone decks often contain multiple cards which act in synergy, e.g. pirate minion cards of the same type buff each other. Generated decks can be categorized in the three major categories: Aggro, Mid-range, and Control. Aggro decks build on purely offensive strategies, which often include a lot of minions. Control decks try to win on the long run by denying the opponent from executing his strategy and dominating the game situation. Mid-range decks are in-between Aggro and Control decks. They try to counter early attacks to dominate the game board with high cost minions during the mid of the game.

Game length and branching factor can be dependent on the decks being played by both players. The possibility of making multiple moves per turn and the enormous amount of possible decks make Hearthstone a challenging problem for AI research.

### 3 Flat Monte Carlo and Monte Carlo Tree Search (MCTS)

MCTS is a heuristic search algorithm commonly used in a wide range of computer game AIs [2]. Its exploration of the game tree consists of four major steps (1) selecting a node, (2) expanding the node with any legal move, (3) simulating (random) playouts called rollout, (4) determining the final value of the playout and propagate it back along the path to the root node. Those four phases are commonly grouped into tree policy (selection and expansion) and default policy (simulation and back-propagation). Moves under consideration can be evaluated by repeatedly simulating games to approximate the players chances of winning the game after executing this move. This can either be done by counting the

number of won simulated games or rating any intermediate simulated game-state using a scoring function. The number of simulations as well as the quality of the playout are crucial in determining an accurate estimate of the chances of success. Finally, the node with the highest success-rate is played.

Using a random selection and expansion as tree policy can lead the agent to lose a lot of time on the simulation of unpromising nodes. For example if nodes are known, of which the first wins in 80% of the computed simulations and the second in only 10 % it would likely be more useful to further analyse the subtree of the first node. This can increase confidence in the approximated chances of winning the game after picking the node. Nevertheless, it would be possible that more simulations on the second node would uncover new and more promising paths. We might also expand a new node, which was not considered yet, that would be even better than the previous nodes. Therefore, the UCB formula (see Equation (1)) [12], which balances exploration and exploitation, can be used during the selection step.

$$\underbrace{R(s')}_{\text{Exploitation}} + C \underbrace{\sqrt{\frac{\log_2(V(s))}{V(s')}}}_{\text{Exploration}} \quad (1)$$

where  $s'$  is a child node of  $s$ ,  $R(s')$  is the average success after choosing node  $s'$ , and  $V(s)$  counts the number of visits of state  $s$  during previous episodes. The constant  $C$  balances both parts of the equation. Using the UCB as a tree policy, the search tree is known to converge to the minimax tree as the number of simulation grows to infinity [12].

## 4 Previous work and the Hearthstone AI competition

Hearthstone is closed source. Thanks to the efforts of an active community, multiple simulators exist as part of the HearthSim project [10].

This work uses a simulator called Sabberstone [5], which tries to remodel each part of the game as close as possible. The C#-programming interface allows researchers to implement algorithms in this rich test environment. Sabberstone currently implements 98% of cards included in the game. Therefore, to the best of our knowledge, Sabberstone represents the most complete simulator currently available. In the future, researchers will be able to directly compare their results in the Hearthstone AI competition [8].

Metastone is another simulator, which was frequently used for research projects in the past [6]. It already includes a greedy optimization agent, which uses a scoring function to choose the best sequence of moves for the current turn. The scoring heuristic takes each player’s minions, their number of hand cards, and current health points into account. Each minion receives a score, which is determined by weighting the minion’s attack and health values, as well as taking typical abilities, such as taunt, into account. The proposed scoring function results in an aggressive play-style, which heavily relies on reducing the opponent’s health,

as well as achieving minion dominance on the game board. Other heuristics were developed for the AAAI17 Data Mining Challenge [11], which inspired multiple research papers on the development of winner-prediction models. Very good results were achieved by Neural Network based methods, which achieved about 80% prediction accuracy of mid-game game-states. Therefore, more enhanced search heuristics than the one provided by Metastone are possible [9].

Metastone also includes an implementation of a Flat Monte Carlo agent, which we use in our evaluation. Based on Metastone, two MCTS agents were developed [13,14], which both performed well against the random and Flat Monte Carlo agent. However, no comparable data is available.

Another well-received work was done on next card prediction [3]. A bag-of-words of card-co-occurrence bi-grams was used for training a prediction system for the next upcoming card. Prediction rates of up to 95% were recorded during the evaluation. The high prediction accuracies inspired our work on enhancements for MCTS.

## 5 Enhancing MCTS by the Prediction of Opponent Hand Cards

Hearthstone players deduce the best move under the effect of multiple sources of uncertainty. Throughout the game the current game-state cannot be fully accessed by the player. During the player’s move it is unknown which card will be drawn next, which cards our opponent currently holds in his hand, and which cards are contained in the opponent’s deck. Even if the game-state would be fully accessible, players need to anticipate their opponent’s next move(s) for laying out their own strategy. Nevertheless, players can elicit different levels of skill, which is suggesting that those tasks can at least be partially handled.

Applying MCTS will have to face the same sources of uncertainty. In this work we predict our opponent’s move from a database of frequently co-occurring cards. Predicting our opponent’s move increases the accuracy of performed simulations and let us reduce the number of necessary simulations without loss of quality [7]. Ultimately, this increases the skill level of the MCTS agent with only limited overhead during the simulation.

### 5.1 Bigram Extraction

Building up a knowledge-base of card-co-occurrences was done by analysing a total of 544.628 replays by human players. The data set was obtained from an openly available database of replays [4]. On this website players are able to upload replays of past games for statistical tracking. Our knowledge-base is based on replays from June 2016 to October 2017. Each replay consists of the players deck as well as a history of moves for the recorded game. Additionally, decks in the replay file are classified in deck categories such as “pirate warrior” or “token paladin”, each consisting of cards which exploit certain card synergies. The amount of games per hero class are summarized in Table 1.

For each card we determined the number of co-occurrences with other cards. Three types of co-occurrences were differentiated

- **isolated:** cards need to be played at the same turn
- **successive:** cards need to be played in successive turns
- **combined:** isolated and successive counts are added

We further studied the influence of taking the games result into account on the final system skill level. For this purpose we either counted only co-occurrences for moves of the winning player, the losing player, or both. The co-occurrence database was stored as compressed .json-file and can be accessed during the simulation phase for to determine likely cards for the next turn.

Table 1: Game statistics

Hero	avg. length	avg. actions	#games
Hunter	393.213	12.927	42.740
Druid	433.383	15.889	72.867
Warrior	428.505	14.757	77.526
Priest	525.571	17.156	53.133
Mage	506.471	18.920	63.887
Shaman	443.933	14.706	76.092
Paladin	459.677	14.561	56.395
Rogue	436.423	17.103	58.541
Warlock	449.750	15.681	43.447
All	452.932	15.786	544.628

## 5.2 MCTS Enhancement

In this work we extend MCTS for creating an agent for Hearthstone. In our adaptations of MCTS we aim for widening the prediction to the next 3 turns determined by consecutive MCTS searches. Extending the simulation can only be allowed by the knowledge-base. The full algorithm is described below. For an easier understanding we recommend comparing the described process with Figure 3.

*Phase 1:* During our MCTS simulation we use the current state of the game board as the initial root node. Each action, such as playing a card, attack with a minion, etc., advances the state of the game and represents another node. The next node is selected based on UCT selection using the Metastone score and the visit count of the node (see Equation (1)). Each transition consists of exactly one move. A player’s turn is made of multiple moves and ends with an end-turn move.

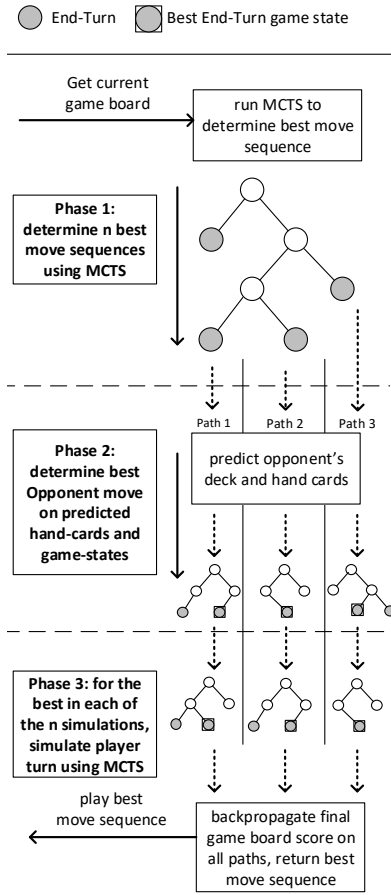


Fig. 3: MCTS with prediction

A simulation uses a greedy selection of moves based on the Metastone scoring function. The last move of a simulation consists of an end-turn move. The final score after ending the player's turn is back-propagated along the simulation path.

*Phase 2:* After a number of simulations we pick the best  $n$  nodes and use the game-state of their best leaf node (end-turn node) for further simulations of the opponent player in phase 2. Based on his previously played cards we determine a likely deck (out of 20 pre-implemented samples). A set of hand-cards is determined using the bi-gram database. For this we use all previously played cards to determine their most probable follow-up card. Out of all follow-up cards we randomly sample the opponent's hand cards to approximate the real set of hand-cards. Using MCTS we determine the best sequence our opponent can play all resulting in a end-turn move and a final game-board.

*Phase 3:* In phase 3, we determine the best follow-up turn for each of the returned game-states to rate the resulting game-board. Once more, this is done using MCTS. The scores are back-propagated all the way to our node-rating in Phase 1. Finally, the best rated node will be picked for execution.

Even if this process turns out to be computationally expensive, the thorough simulation of our opponent assures a high quality prediction during phase 2. Deeper simulations can be created by stacking phase 2 and 3 multiple times, before back-propagating the final game-board scores. Nevertheless, we chose a depth of 3, because of the limited prediction accuracy for follow-up turns. Simulating deeper playthroughs would accumulate more uncertainty, due to unknown card-draws on both sides. It turned out to be more effective to increase the number of simulations for a better approximation of the game-state score after a few moves. The number of simulations can be adjusted to fit the 75 seconds time limit of the game.



## 6 Evaluation

### 6.1 Experimental Setup

During our evaluation we tested the proposed agent against multiple other agents mentioned in the literature or in previous implementations. We included a random agent (random) as general baseline without any planning capabilities. Flat Monte Carlo (flatMC) was implemented as a basic search strategy. MCTS without further prediction of the opponent’s hand cards (plainMCTS) was implemented to test the influence of our bigram prediction. Full observation MCTS (foMCTS) was used to compare our AI versus an optimal prediction of opponent’s hand cards. Here, we simulate the optimal prediction by providing the AI with the actual hand cards during the current game-state. Finally, an exhaustive search (exh. s.), which is implemented by the Sabberstone framework, allowed us to choose the best of all possible move sequences for the players turn. However, the method is not allowed to simulate the opponent’s turn.

Our proposed MCTS algorithm with prediction of our opponent’s hand cards (predMCTS) was tested against each of the described agents. Therefore, we used three by Sabberstone pre-implemented decks for representing the possible deck types and play-styles in Hearthstone. The “Aggro Pirate Warrior” deck (Aggro) consists of low cost pirate minions, offensive weapons, and spell cards of the warrior class. In contrast, the “Mid-Range Jade Shaman” deck (Mid-Range) is made of multiply minion buffs and damaging spells. The third deck-type (Control) is represented by the “Reno Kazaku’s Mage” deck, which is mainly based on spell damage, and destroying or taking over control of enemy minions.

For each combination of player deck, opponent deck, and opponent agent we simulated at least 100 games. The multiple random factors of hearthstone lead us to simulate more games for the comparison of predMCTS versus plainMCTS and foMCTS to get stable results. Winning percentages for each match-up are summarized in Table 2.

### 6.2 Discussion

Reviewing the results in Table 2 indicates the advantages of the implemented AI approach. Looking at all the match-ups in which both agents play the same deck, it becomes clear that our proposed agent is able to beat them in all cases except one. In general the agent performs best when playing the Mid-Range deck. Playing the Aggro deck also leads to very good results. The results of the Control deck against other decks are worse than the results of both other decks, which might be due to the deck being generally worse than the other two variants.

Both random and flat Monte Carlo consistently lose in most match-ups, while the remaining opponents are able to win games more often. It is not surprising that nearly all games ( $\approx 98\%$ ) are won against the random player. Games against the flat Monte Carlo agent show a promising success rate of  $\approx 81\%$  on average.

The plainMCTS agent was beaten in most games while playing the Mid-Range or the Aggro deck. However, our AI performs worse in the match-ups where

Table 2: Winning chances of predMCTS using various decks against other agents. 100 games were simulated against the Random, flatMC, and exhaustive search agent. Up to 500 games were simulated against predMCTS and foMCTS, because bigger variances were observed during the simulation process. Columns in which both agents play the same deck are highlighted in gray.

Wins in %	Aggro	Mid	Control	Wins in %	Aggro	Mid	Control
Random	95	100	100	Random	99	98	100
flatMC	81	73	94	flatMC	88	85	99
plainMCTS	59	47	58	plainMCTS	71	55	76
foMCTS	46	36	60	foMCTS	59	50	76
exh.s.	65	47	70	exh.s.	62	70	85

(a) predMCTS Aggro Deck

(b) predMCTS Mid-Range Deck

Wins in %	Aggro	Mid	Control
Random	97	97	100
flatMC	73	54	89
plainMCTS	36	31	68
foMCTS	41	16	51
exh.s.	45	20	61

(c) predMCTS Control Deck

it uses the Control deck against other deck types. Nevertheless, we achieved a win-rate of 68% in cases where both agents play the control deck. The overall win-rate is  $\approx 56\%$ . This result and the better performance in the other match-ups where both agents play the same deck shows that the improved prediction of the opponent move results in a better overall performance.

The only agent that has beaten our proposed approach in more than 50% of simulated games is the foMCTS agent. Our overall win-rate is  $\approx 48\%$ . This is not surprising due to the fact, that the foMCTS receives full information of the players hand cards. Therefore, our results indicate that precise knowledge can increase the performance of the MCTS algorithm by a large degree. For this reason, we are motivated to even further increase the performance of our card prediction algorithm. The advantage of knowing the true hand cards is a bit unfair and impractical in a legal game situation.

Our proposed approach was able to beat exhaustive search in 6 out of 9 match-ups. The overall win-rate is  $\approx 58\%$ .

Generally, the player playing the control deck performs worse in most match-ups. This could be due to two possible reasons. First, short-term prediction is not suitable for playing a control deck, because the used deck is in need of long-term planning. Therefore, the implemented approaches may not perform very well. A planning horizon of more than three turns may increase the play-strength

on this deck-type. Another reason may be Metastone’s scoring function, which puts more weight on the current game-board, than on the remaining utility of the hand cards effects. Playing a spell that kills all the opponent’s minions can be played for immediate effect or kept for a later situation. For example killing just one minion would be a waste, if another card could yield the same effect. Either improving Metastone’s scoring function or implementing a deck dependent scoring function may increase the play-strength for this deck.

## 7 Conclusions and Future Works

In this work we proposed a method for handling uncertainty in MCTS. Our method involves the creation of a knowledge-base, which is used to initialize an ensemble of MCTS procedures. Our current sampling approach is based on bi-grams, but can theoretically be exchanged by any probabilistic or heuristic sampling. The resulting decision-making based on such an ensemble proved to be less prone to uncertainty at the cost of a marginally increased computation time. Our sampling based ensemble opens up a new class of MCTS algorithms, which in its current version already outperforms other agents based on MCTS.

We evaluated our approach based on the collectible online card game Hearthstone. Bigrams, which are fast to process and cheap in memory consumption, are learned from a database game replays. The simulation phase is guided by sampling multiple hand card sets for the opponent player based on the gathered knowledge-base. Due to this sampling multiple game situations are considered during the decision-making phase. This enables our agent to choose better moves than comparable agents depending on the current state of the game. The proposed agent was able to consistently beat those agents in multiple game setups.

Our tests using an MCTS agent with full information on the current game-state show that a perfect prediction would yield slightly better results in some match-ups, suggesting that improving the prediction accuracy would even further increase the play-strength. In the future we plan to further analyse the capabilities of partially informed MCTS agent ensembles.

In the context of creating a Hearthstone AI, we plan to further extend the opponent card and deck prediction. For this purpose, we would like to incorporate other sources of knowledge, such as a deck database of current meta-decks. Further adaptations need to be made for detecting commonly played decks and inferring the opponent’s strategy. In this work we limited our deck database to 20 commonly played decks and the deck strategies Aggro, Mid-Range, and Control. This can be further extended by focusing on deck dependent core-mechanics, such as strong board clears, discarding cards, or buffing minions. Detecting which core-mechanic the deck relies on would further improve the opponent prediction. Also, updates of cards or additions of new cards can drastically change the meta-game. Hence, gathered knowledge needs to be frequently revised to stay up to date with currently played strategies. Further work will be put into better scoring functions, which are based on those core-mechanics.

## References

1. "Blizzard Entertainment": Hearthstone webpage, <https://playhearthstone.com/en-gb/0>, accessed on 06.03.2017
2. Browne, C.B., Powley, E., Whitehouse, D., Lucas, S.M., Cowling, P.I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., Colton, S.: A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in Games* 4(1), 1–43 (mar 2012), <https://doi.org/10.1109/TCIAIG.2012.2186810>
3. Bursztein, E.: I am a legend: Hacking hearthstone using statistical learning methods. In: 2016 IEEE Conference on Computational Intelligence and Games (CIG). pp. 1–8. IEEE (sep 2016), <https://doi.org/10.1109/CIG.2016.7860416>
4. "Collect-o-Bot": Hearthstone replay database, <http://www.hearthscry.com/Collect0Bot>, accessed on 06.03.2017
5. "darkfriend77": Sabberstone github repository, <https://github.com/HearthSim/SabberStone>, accessed on 06.03.2018
6. "demilich1": Metastone github repository, <https://github.com/demilich1/metastone>, accessed on 06.03.2017
7. Dockhorn, A., Doell, C., Hewelt, M., Kruse, R.: A decision heuristic for Monte Carlo tree search doppelkopf agents. In: 2017 IEEE Symposium Series on Computational Intelligence (SSCI). pp. 1–8. IEEE (nov 2017), <https://doi.org/10.1109/SSCI.2017.8285181>
8. Dockhorn, A., Mostaghim, S.: "Hearthstone AI Competition", <http://www.is.ovgu.de/Research/HearthstoneAI.html>, accessed on 06.03.2018
9. Grad, L.: Helping AI to Play Hearthstone using Neural Networks. In: Ganzha, M., Maciaszek, L., Paprzycki, M. (eds.) *Proceedings of the 2017 Federated Conference on Computer Science and Information Systems*. vol. 11, pp. 131–134 (sep 2017), <https://doi.org/10.15439/2017F561>
10. HearthSim Project: Hearthsim webpage hearthstone simulation & ai, <https://hearthsim.info/>, accessed on 06.03.2017
11. Janusz, A., wiechowski, M., Zieniewicz, D., Stencel, K., Puczniewski, J., Madziuk, J., Izak, D.: Aaia'17 data mining challenge: Helping ai to play hearthstone, <https://knowledgepit.fedcsis.org/contest/view.php?id=120>, accessed on 06.03.2017
12. Kocsis, L., Szepesvári, C.: Bandit Based Monte-Carlo Planning. In: Fürnkranz, J., Scheffer, T., Spiliopoulou, M. (eds.) *ECML'06 Proceedings of the 17th European conference on Machine Learning, Lecture Notes in Computer Science*, vol. 4212, pp. 282–293. Springer Berlin Heidelberg, Berlin, Heidelberg (2006), [https://doi.org/10.1007/11871842\\_29](https://doi.org/10.1007/11871842_29)
13. Santos, A., Santos, P.A., Melo, F.S.: Monte Carlo tree search experiments in hearthstone. In: 2017 IEEE Conference on Computational Intelligence and Games (CIG). pp. 272–279. IEEE (2017), <https://doi.org/10.1109/CIG.2017.8080446>
14. Tzourmpakis, G.: Hearthagent, a hearthstone agent, based on the metastone project, <http://www.intelligence.tuc.gr/~robots/ARCHIVE/2015w/Projects/LAB51326833/>, accessed on 06.03.2017