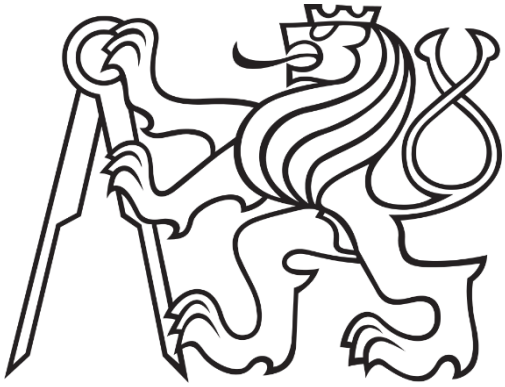
Czech Technical University in Prague

Faculty of Electronical Engineering

Department of Computer Graphics and Interaction



Bachelor thesis

Artificial Intelligence Methods for Playing Collectible Card Games

Patrik Březina

Supervisor: MGR. Viliam Lisý, MSC., PH.D

Artificial Intelligence Methods for Playing Collectible Card Games

BACHELOR'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Březina Patrik** Personal ID number: **466244**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Computer Graphics and Interaction**

Study program: **Open Informatics**

Branch of study: **Computer Games and Graphics**

II. Bachelor's thesis details

Bachelor's thesis title in English:

Artificial Intelligence Methods for Playing Collectible Card Games

Bachelor's thesis title in Czech:

Metody umělé inteligence pro hraní sběratelských karetních her

Guidelines:

Collectible Card Games are a popular segments of games that human play for recreation as well as professionally. The student will:
1) Extensively review the existing literature on creating AI players for these games;
2) review and select an existing open source implementation of a specific collectible card game;
3) port or re-implement at least two substantially different approaches to creating AI in this game;
4) rigorously compare the quality of these approaches and its dependence on their main parameters.

Bibliography / sources:

[1] Zhang, Shuyi. 'Improving Collectible Card Game AI with Heuristic Search and Machine Learning Techniques.' PhD diss., University of Alberta, 2017.
[2]D. Churchill and M. Buro, "Hierarchical portfolio search: Prismata's robust AI architecture for games with large search spaces," in Proceedings of the Artificial Intelligence in Interactive Digital Entertainment Conference, 2015.
[3]C. D. Ward and P. I. Cowling, "Monte Carlo search applied to card selection in Magic: The Gathering," in Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on. IEEE, 2009, pp. 9–16.
[4]Stiegler, Andreas, Keshav P. Dahal, Johannes Maucher, and Daniel Livingstone. 'Symbolic Reasoning forHearthstone.' IEEE Transactions on Games 10, no. 2 (2018): 113-127.

Name and workplace of bachelor's thesis supervisor:

Mgr. Viliam Lisý, MSc., Ph.D., Artificial Intelligence Center, FEE

Name and workplace of second bachelor's thesis supervisor or consultant:

Date of bachelor's thesis assignment: **24.01.2019** Deadline for bachelor thesis submission: _____

Assignment valid until: **20.09.2020**

Mgr. Viliam Lisý, MSc., Ph.D. Supervisor's signature	Head of department's signature	prof. Ing. Pavel Ripka, CSc. Dean's signature

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guidelines for adhering to ethical principles when elaborating an academic final thesis.

In Prague on May 24.05.2019 _____

Acknowledgements

I would like to express my gratitude to my supervisor Viliam Lisý who offered consultation, guidance and encouragement during the development of my thesis.
Also, I would like to thank my family for all the moral support they offered, motivating me to keep pushing forward. I would like to express special thanks to my brother for lightening up my mood whenever I felt miserable, my father for his assistance with various technical issues encountered throughout my studies and for helping me to dive into the world of programming, and my mother for taking care of me for all those years.
Access to computing and storage facilities owned by parties and projects contributing to the National Grid Infrastructure MetaCentrum provided under the programme "Projects of Large Research, Development, and Innovations Infrastructures" (CESNET LM2015042), is greatly appreciated.

Artificial Intelligence Methods for Playing Collectible Card Games

Abstrakt

Sběratelské karetní hry jsou populární hry, které lidi hrají nejen pro zábavu, ale i na profesionální úrovni. Představují řadu výzev pro implementaci umělých hráčů, mezi které patří nejistota způsobena náhodnými událostmi, vysoké větvení rozhodovacího stromu nebo skrytá informace reprezentována neznámým obsahem protivníkova balíčku a ruky. V této práci jsou zmíněny přístupy zabývající se touto problematikou a dva z nich jsou reimplementovány a porovnány proti sobě v simulátoru Metastone, jenž implementuje logiku sběratelské karetní hry Hearthstone: Heroes of Warcraft, která je použita jako testovací prostředí.

Klíčová slova:

Umělá Inteligence
Sběratelské Karetní Hry
Hearthstone
Monte Carlo Tree Search
Neurální síť
Strojové Učení

Artificial Intelligence Methods for Playing Collectible Card Games

Abstract

Collectible Card Games are popular games played by people for recreation as well as professionally. These games represent number of challenges for implementation of artificial players, such as uncertainty caused by random events, high branching factor of decision tree or hidden information represented by unknown content of opponent's deck and hand. In this thesis, various works addressing development of artificial player for collectible card games are introduced. Two selected approaches are then reimplemented and compared against each other in the Metastone simulator which implements logic of collectible card game Hearthstone: Heroes of warcraft that is used as a testbed for the conducted experiments.

Keywords:
Artificial Intelligence
Collectible Card Games
Hearthstone
Monte Carlo Tree Search
Neural Network
Machine Learning

Contents

List of Figures

List of Tables

Terminology

CCG	Collectible Card Game
M:TG	Magic: The Gathering
NN	Neural Network
EA	Evolutionary Algorithm
GSV	Game State Value
MCTS	Monte Carlo Tree Search
UCB	Upper Confidence Bound
UCT	Upper Confidence Bound Applied to Trees
PIMC	Perfect Information Monte Carlo
ISMCTS	Information Set Monte Carlo

1 Introduction

With the growth of the game industry grows the expectations and demands on the quality of the games. One of the game's qualities can be offered challenges and how realistic the game feels. Both these aspects can be addressed with the usage of artificial intelligence, be it an intelligence of a non-player character in role-playing game who works on field during daytime and attempts to find shelter when it starts raining, creating more realistic environment of the game, or an opponent who attempts to mimic behavior of an experienced player, challenging the player who plays the game. In recent years, development of artificial intelligence achieved remarkable success in classic video games, Go or Poker, defeating professional players. However, some other popular games such as collectible card games have not been thoroughly explored yet. Their complex rules, frequent random events, high branching factor of decision tree and unknown opponent's deck that contains only a fraction from thousands of possible cards make these games difficult to model and this creates a challenging environment for developing an artificial intelligence.

2 Thesis Goals

The goal of the thesis is to examine and discuss existing approaches to development of artificial players in the collectible card games that address various challenges presented by these games. Then to choose a specific collectible card game and review its available open-source implementations and finally reimplement selected approaches, discuss their parameters and compare their quality.

3 Background

3.1 Collectible Card Games

This section introduces collectible card games and describes what challenges these types of games represent for the development of artificial intelligence.

3.1.1 Description

Collectible card games (CCGs) are strategic card games that consist of specially designed sets of playing card. Each playing card is described with an image and a text which defines the effects of the card when it is played. Games are typically played between two players, however there are also multiplayer formats (such as Commander format in Magic: The Gathering). Each player constructs their own decks from the set of available cards (hundreds or even thousands of cards) with which they play. Decks are unknown to the opponent and therefore represent hidden information that is challenging to deal with. Collectible card games became popular with the release of Magic: The Gathering (M:TG) by the company Wizards of the Coast in 1993 and new sets of playing cards are still produced. CCGs enjoyed another great rise in popularity with the release of free-to-play game Hearthstone: Heroes of Warcraft in 2014 developed by Blizzard Entertainment. Success of these games inspired many developers to design their own collectible card games such as Yu-Gi-Oh! (developed by Konami), Pokémon Trading Card Game (Media Factory), Eternal (Dire Wolf Digital), Gwent (CD Projekt Red) and many others.

3.1.2 Challenges

- **Imperfect information**

  In collectible card games each player constructs a deck from cards they own. This can include hundreds or even thousands of different unique cards each with their own effects. Decks typically have some limitation for how many cards it can or must contain (in M:TG decks have to consist of 60 or more cards, in Hearthstone it is exactly 30) Players do not know what cards the opponent's deck contains. Determinizing the opponent's deck can be very beneficial for the player as it allows them to adjust their strategy accordingly. However, each deck from which the players

draw their hands are randomly shuffled and even if the player knew what cards the opponent's deck consists of, they have no information about what the opponent is currently holding in hand. Also, depending on the effect some cards can be played face down and their effect does not execute until a certain event has occurred (such as secret cards in Hearthstone or cards with Morph in M:TG). Each card played by the opponent gives away information that can be used to determinize the remaining hidden information. Correct determinization of unknown data could increase player's chances of winning the match. On the other hand, guessing the information incorrectly could have opposite effect and hinder their performance.

- **Chance events**

  Players draw cards from their deck which is randomly shuffled at the beginning of the match. Therefore, it is difficult to predict what cards the player will hold in the coming rounds. This complicates planning approaches as it would be necessary to plan for all the possible outcomes of card drawing. For example, in M:TG each deck has to contain at least 60 cards. First card draw from the deck has 60 possible outcomes and it would be difficult to plan for all the possibilities. Furthermore, effects of some cards can cause random events during the gameplay such as choosing a random target, placing a random card on the board, discarding random card from player's hand or the execution of the effect can depend on a coin flip. All this can lead to unpredictable and improbable game states. Therefore reduction of the branching factor in chance events is necessary but omitting too many possible outcomes might lead to badly informed decision-making.

- **Game complexity**

  Each card has its own effect and some of those effects can even change rules of the game during match. Some effects can seem to be weak at first, but in combination with other cards its power can be multiplied. This creates synergies between the cards and certain combinations of cards can lead to very explosive effects (such as never ending turn, killing opponent in one turn, having infinite amount of health). While constructing a deck it is important to consider these possible interactions between cards and use them to one's advantage. Artificial intelligence could be utilized to detect these synergies between cards. Not only it would help the player to construct a strong deck, but it could also assist the game's designers to detect possible synergies of newly designed cards and adjust their power accordingly or reconsider

their addition to the game. Also, being able to recognize synergies between cards can be exploited to create more accurate determinization of the opponent's deck because opponent's deck is likely to contain cards that synergize with each other.

- **Strategic game approach**

  During the gameplay of CCGs players alternate in turns and can perform multiple actions until they decide to end the turn on their own. Players typically have limited amount of resources (lands in M:TG, mana crystals in Hearthstone, but also cards in hand or on board) which limits the number of actions players can take in their turn. To plan their turn optimally they need to consider each resource they have available and use it at its maximal potential. Players can play cards from their hands or interact with cards they have placed on the playing board by using their effects and attacking with their units. They can switch between board interaction and card playing as they like but can also decide to end their turn whenever they want, even if there are still some available actions to preserve their resources for the future or to not give information away to the opponent. Not only it is necessary to plan for the current turn but to also plan for the future turns which can prove time consuming for an artificial player.

3.2 Hearthstone

This part describes the functionality and gameplay of collectible card game Hearthstone: Heroes of Warcraft that is used as a base for the experiments. This card game is popular among people and its ruleset is significantly reduced compared to other CCGs like M:TG, therefore Hearthstone is utilized as testbed for majority of research regarding CCGs.

3.2.1 Game Description

Hearthstone is zero-sum collectible card game with imperfect information that is played between two players who alternate in turns. Before the match each player selects one of nine available heroes. Each hero has their own Hero Power - a special action different for each hero that can be used once a turn (for example Priest can heal any target for 2 health), and their own set of cards. Cards of the selected hero can be combined with neutral cards that are available to all heroes to construct a deck with exactly 30 cards.

At the beginning of a match a coin flip decides which player goes first. Player who goes first draws 3 cards in their starting hand and the second player draws 4 cards and gets a special card called "The Coin" that can be used to gain 1 additional mana crystal until end of turn. Drawing of starting hand is followed by mulligan phase. In mulligan phase each player selects any number of cards in their starting hand. Each selected card is then shuffled back into the player's deck and is replaced with another card from the deck. This gives players some control over the contents of their initial hand.

Every hero has 30 health at the start of the match and the goal of the game is to reduce opponent's health to zero. At the beginning of their turn players draw a card from their deck, gain 1 additional mana crystal up to a maximum of 10 and all depleted mana crystals are refreshed. Hand of each player is limited to 10 cards and if player draws a card while having 10 cards in hand the drawn card is discarded. If a card is drawn while there are no more cards left in the player's deck the player draws a "Fatigue" card instead. Fatigue cards ignore hand size limit and are immediately played after drawing, dealing damage to the player according to the number of "Fatigue" cards drawn in total (first Fatigue deals 1 damage, second deals 2, third deals 3 and so on).

During their turn players can perform various actions:

- **Play card**

  Playing a card from players hand to perform the card's effect. Every card has its own manacost that describes how much mana crystals have to be depleted to play the card. Depleted mana crystals are disabled for the rest of the turn.

- **Card selection**

  Player can place minion cards on board and equip their hero with weapons. Minions and equipped heroes can be selected to attack one of the opponent's minions or the opponent.

- **Target selection**

  When attacking or playing certain cards from hand valid target needs to be selected. Valid targets for attacking are opponent's hero and minions they control. However, targeting restrictions for card's effects depend on the played card. Some cards can target anything present on the board, other card can only target units controlled by the opponent or the player.

- **End turn**

  Anytime during their turn players can choose to end it and give initiative to the opponent. Players aren't forced to execute all available actions and can end their turn whenever they decide to.

Whenever health of any player is reduced to 0 the game ends and the surviving hero wins. If health of both heroes reaches 0 at the same moment the game ends in a tie.

3.2.2 Game Components

This part describes entities in a game of Hearthstone.

3.2.2.1 Game Board

Game board is a component that represents the current state of the game. It contains all observable information visible to the player. The game board and its components in the game are described in figure 1. The Game board is represented by:

- **Heroes**

  Two heroes are on the board, each representing one player. Each hero has their own health, armor (damage is taken from armor before health), weapon, hero power, mana crystals and secrets (specific cards played faced down represented as a question mark). If the players wish to select opponent or themselves as target they select the hero who represents them.

- **Battlefield**

  Battlefield is an area where players place their minion cards. It is split in half, one for each player. Players can have up to 7 minions placed on their half of the battlefield. Minions placed on the battlefield can be selected for attack actions as well as targets of other cards.

- **Decks**

  Each player draws cards from their deck that consists of exactly 30 cards. Players can't see the remaining content of either deck, only the information about the quantity of remaining cards is available.

- **Hands**

  Each player has their own hand that can contain up to 10 cards. The player's hand is visible to them but hand of the opponent is hidden. However, players can see how many cards their opponent has in their hand.

Figure 1 – Game Board (source Hearthstone)
1-Player's hero, 2-Opponent's hero, 3-Hero power, 4-Player's hand,
5-Opponent's hand, 6-Player's mana crystals, 7-Player's deck, 8-Opponent's deck,
9-Opponent's half of the battlefield, 10-Player's half of the battlefield

3.2.2.2 Cards

In Hearthstone each card consists of its artwork, text description of its effect, manacost that defines how much mana crystals have to be spent to play the card and, if the card is a minion or a weapon, also a numeric description of its health and attack (Figure 2). There are 4 types of cards in Hearthstone:

- **Minions**

  Minions are cards placed on the battlefield. Each minion has its own attack and health. Attack defines how much damage the minion deals to the target when

attacking and to the attacker when being attacked. How much damage the minion can take is based on health and when minion's health is reduced to 0 it leaves the battlefield. Each minion can also have its own effects. For instance, Flame Juggler (Figure 3.a) deals 1 damage to a random enemy when played, Flametongue Totem (Figure 2) adds 2 attack to minion on its left and on its right while it remains on the board.

- **Weapons**

  Like minions, each weapon has its attack value and health. Weapons are cards that are equipped on heroes. Hero can have only 1 weapon equipped and playing a weapon while another one is equipped will destroy the attached weapon and replace it with the new one. During player's turn the hero gets attack power equal to the attack of the equipped weapon and is allowed to attack opponent's hero or minions to deal the weapon's damage to the target and take damage equal to the defender's attack value. Each attack with the hero decreases health of the weapon by 1 and when its health reaches 0 the weapon breaks. Weapons can have their own effects. Death's Bite (Figure 3.b) has 4 attack power, 2 durability and deals 1 damage to all minions on the battlefield when it is destroyed.

- **Spells**

  Spell are cards that cause an immediate effect when played and leave the game after the effect is done. Some spells might require the player to select a target, such as Fireball (Figure 3.c) that deals 6 damage to any target, while other spells might not need a target at all, for example Arcane Intellect draws 2 cards to the player who played it or Flamestrike that deals 4 damage to all minions on opponent's side of the battlefield.

- **Secrets**

  According to game's logic secrets are also considered spells (this is important for cards that interact with spell cards), but unlike spells, secrets don't have an immediate effect on the game board. Instead they are attached to the hero face down marked as a question mark to the opponent and wait for a triggering event. When triggered, secrets execute their effect and leave the game afterwards. Counterspell (Figure 3.d) is a mage secret card that triggers when opponent plays a spell. If they do, Counterspell is triggered and the effect of the played spell is negated, doing nothing as a result.

Figure 2 – Description of Flametongue Totem (source Hearthstone)

1-Manacost, 2-Attack value, 3-Health, 4-Card's effect

3.a) minion 3.b) - weapon 3.c) - spell 3.d) – secret

Figure 3 – Hearthstone card examples

3.3 Summary

In this chapter the Hearthstone collectible card game, its mechanics and how the game is played was introduced. Then, various components of the game including the game board, construction of player's deck or different types of cards and their functionality were discussed.

4 Hearthstone Simulators

This section discusses available open-source simulators for Hearthstone and functions they offer. Afterwards the simulator Metastone that is used as a testbed for experiments in this work is introduced.

4.1 Fireplace

Fireplace [1] is a simulator created by community HearthSim [2]. Fireplace loads available cards from the Hearthstone's game files and offers an API to manually change cards or define new ones. Simulator to play series of games between two artificial players is available as well, however it is not possible for the user to play against the agents on their own and the simulator offers no graphical user interface.

4.2 Hearthbreaker

Hearthbreaker [3] is another simulator developed by community Hearthsim [2] designed for machine learning and data mining purposes. It offers a wide variety of cards and a simple simulator that can be used to determine synergies between cards, but it does not support simulation of games between two players.

4.3 HearthSim

HearthSim simulator [4] is designed for simulating a desired number of matches between two artificial players with graphical overview of the results, computing average win rate and confidence interval of win rate with confidence level of 95%. However, simulator offers very limited pool of cards.

4.4 Hearthstone++

Hearthstonepp [5] offers interface for simulations between two artificial agents in console and in available graphical user interface, support for implementation of reinforcement learning algorithms and about 50 different cards for deck building.

4.5 Metastone

Metastone [6] is an open-source Hearthstone simulator that credibly replicates the flow of the Hearthstone game. It offers a graphical user interface with a deck builder to create new decks from about 1300 available cards, a game mode where the user can play against artificial players as well as observe gameplay between two bots or a simulation mode to simulate number of games between two artificial players with results overviewed at the end of the simulation. Results give an information about the win rate of each individual player, number of played cards, total damage taken during all simulations, most played card and more. However, Metastone does not support simulation of games through console which proved problematic when attempting to setup tasks in MetaCentrum computational cluster to conduct experiments. This missing feature was added to the simulator.

4.5.1 Metastone AI players

Metastone offers a variety of artificial agents that are capable of playing the game. These agents can be used to test performance of new artificial players.

- **Random agent** selects available actions and targets at random.
- **No Aggression** agent never performs any attack actions and only plays cards randomly from its hand. Unless the players kill themselves, it is impossible to lose against this agent.
- **Do Nothing** always ends turn without performing any action.
- **Greedy Optimize Move** utilizes weighted heuristic to assign scores to each available action based on the game state and selects the one with the highest score.
- **Greedy Optimize Turn** assigns scores to each available action using alpha-beta pruning algorithm that is driven by the same weighted heuristic as Greedy Optimize Move and returns the action with the highest score.
- **The Game State Value** uses an alpha-beta pruning algorithm to assign scores to available actions that is driven by a threat-based heuristic with weights optimized using an evolutionary approach. The heuristic counts a threat level based on the current state of the board. Goal is to minimize threat level of the opponent and maximize player's threat towards the opponent. The move with the highest score is selected. This is the strongest artificial player available in Metastone simulator.

- **Flat Monte Carlo** for each available action number of random simulations is played until the terminal state of the game is reached. The action is then scored according to the number of won simulations and the action with the most wins is returned.

4.5.2 Metastone Game System

Metastone implements complete logic of the game of Hearthstone. GameContext class contains all the information about the current state of the game as well as the gameplay logic for action execution. Each player has their own deck, hand, minions they control, behavior parameter that describes assigned artificial player and hero who is tied with available mana crystals, health, armor, equipped weapon and played secrets. After the initialization of GameContext the game begins. The player who goes first is selected randomly with a coin flip and is marked as an activePlayer. The game is played in cycles until health of one of the players reaches 0. In each cycle the activePlayer is requested to perform an action from a list of valid actions offered by the GameContext. Type of the action is determined by an enum parameter. If an end turn action is returned the current player gives the initiative to their opponent and opponent begins their turn.

Each unique card is saved as a json file. The content of these files includes information about the card's name, its manacost, type of the card (minion, spell, …), description of the card's effect (type of the effect, whether it requires a target, what a valid target is, effect's values) and, if the card is a minion or a weapon, also an attack power and health values. Metastone supports only some of the selected card effects because in real Hearthstone the card effects are diverse and it is difficult to generalize them. Therefore, if the card's effect cannot be described with the defined rules in a json file it is possible to implement the Play() function for the card to perform the card's effect. Each available deck is also described in a json file by its name, hero and collection of contained cards.

4.6 Summary

This chapter presented some of the available open-source simulators for Hearthstone and briefly described each of them. Lastly, the Metastone simulator which is utilized for the experiments conducted in this work, its artificial players and its game system are discussed.

5 Artificial intelligence techniques

This chapter discusses artificial intelligence techniques that are used in this work to design artificial players.

5.1 Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) [7] is a family of search methods designed to address sequential decision-making problems. The method is based on analysis of the most promising nodes and constructs the search tree during its execution, starting in the current state where a decision is required. Each node in the tree is scored according to simulations between AI-controlled players who play moves (randomly or driven by a quick decision-making process such as heuristic) until they reach the terminal state of the game, using the game's result to score the nodes. With each iteration the most promising nodes (nodes that lead to the most victories) are expanded with new nodes, constructing an uneven tree. The method is suitable for implementation of artificial players in games with high branching factors such as Go [8] and its stochastic nature of the simulations allows the method to handle randomness as well.

The MCTS search algorithm consists of 4 phases:

1. **Selection:** In this phase selection function is applied recursively to traverse the tree in order to find a new node to expand (a leaf node or a node not fully expanded). The goal of the traversal in the selection function is to balance between exploitation (the best action so far) and exploration (action that hasn't been thoroughly explored) based on the available information from simulations.
2. **Expansion:** One or more children of the selected node are added to the search tree. Children are states reachable from the selected node.
3. **Simulation (rollout):** For each expanded node one or more simulations between two AI-controlled players are played until a terminal state or a defined depth is reached. The games can be played randomly or be guided by a quick decision-making process. The nodes are then scored based on the game's result (typically +1 if the player won the game, +0 otherwise).
4. **Back-Propagation:** Results of rollouts are propagated to all nodes that were visited from the root state along the path to the newly added node.

5.1.1 UCB – Upper Confidence Bound

As was mentioned in the description of the simulation phase, its goal is to achieve a balance between exploration and exploitation of the tree. A commonly used policy to achieve this balance is Upper Confidence Bound (UCB) [9]. Each node of the tree contains two information:

1. The number of times the node was visited in all the carried simulations
2. The number of victorious simulations from the node.

This information is used to guide the tree traversal based on the following formula:

$$\frac{W(n)}{V(n)} + c\sqrt{\frac{\ln V(p)}{V(n)}}$$

Where W(n) is the number of victories achieved in node *n*, V(n) is the number of visits of node *n*, p is the parent node of node *n* and *c* is an exploration parameter that balances selection between the most successful and less explored nodes. For UCT (Upper Confidence Bound applied to Trees) the exploration parameter is selected empirically. This is computed for each child node of current node and the one with the highest value is selected. With infinite amount of simulations from each node MCTS is guaranteed to converge to minimax tree when utilizing UCB [10].

5.2 Neural Network

Artificial neural networks [33] are computing systems inspired by the biological neural networks and are intended to replicate learning processes of human brain. The goal of an artificial neural network is to find patterns among the training data and use the acquired knowledge to recognize between the entities it was trained for. Artificial neural networks have been used on a variety of tasks including speech recognition, image recognition or game playing.

Artificial neural network is based on entities called neurons. Each neuron takes number of input signals and computes a weighted sum of the inputs, generating an output signal value. The activation equation looks like this:

$$a' = \sigma(w_1 a_1 + w_2 a_2 + \cdots + w_n a_n - b)$$

Where a' is activation value of the neuron, a_n is input value from other neuron, w_n is weight of the neuron, b is bias (activation threshold) and $\sigma(x)$ is an activation function.

The neural network is structured into layers – an input layer that receives encoding of the initial input to be evaluated, an output layer which returns results of the evaluations (output signals) and hidden layers that process the input values. Figure 4 depicts a fully-connected network with 6 input values, 2 hidden layers and 1 output value.

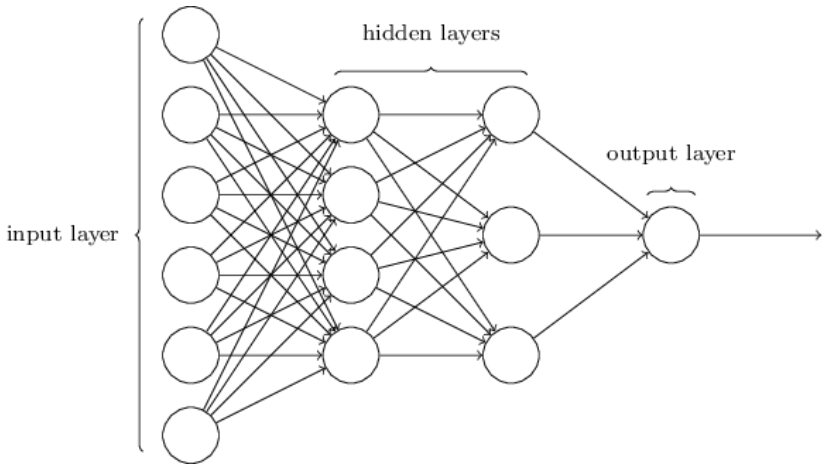


Figure 4 – Fully-connected feed-forward neural network. Adopted from [33]

To teach neural network a set of training data is required where each individual input is also labeled with the expected output. Once the batch of training data is evaluated the network's output is compared to the expected output. A cost function calculates how incorrect the evaluation is compared to the expectation. The goal of the learning process is to minimize the cost function of the output. To achieve this goal, gradient descend is utilized, moving in the opposite direction of the gradient to move towards the minimum. However, the cost function can depend on thousands of variables, making it difficult to calculate its gradient. For this objective a backpropagation algorithm was designed. This algorithm walks backwards through the neural network, propagating calculated errors to the neurons and determining how the parameters influencing the neuron's activation should change to achieve descend towards minimum of the cost function.

5.3 Summary

In this chapter Monte Carlo Tree Search, a method designed for decision-making problems, was introduced. Afterwards the policy called Upper Confidence Bound to balance the exploration and exploitation of the MCTS was discussed and finally artificial neural network and a brief summary of its functionality were presented.

6 Artificial Intelligence Approaches in CCGs

This chapter presents several works related to creation of artificial players in collectible card games. Because of Hearthstone's high popularity it was used as a testing environment for majority of researches related to collectible card games.

6.1 MCTS Enhanced with Domain Knowledge and Heuristics

In [19] a modified Monte Carlo Tree Search utilizing expert knowledge to deal with unknown information and heuristic driven rollouts is proposed as a card playing agent in Hearthstone. The approach modifies selection and rollout phases of the MCTS:

Selection

In order to achieve balance between exploration and exploitation the upper confidence bound is applied during the selection phase. To further refine this balance expert knowledge is added to the UCB formula via progressive bias [20] in form of heuristic:

$$\frac{W(n)}{V(n)} + c\sqrt{\frac{\ln V(p)}{V(n)}} + \frac{H(p)}{1+V(p)}$$

Where H(p) is heuristic value of the parent node returned by heuristic function and V(p) is number of parent node visits.

Rollout

Two modifications are applied to the rollout phase. In simulation phase the games are played until terminal state is reached. To dealt with hidden information a deck database of premade decks is utilized. Based on cards played by the opponent a deck is selected from the database with the most co-occurrent cards and is assigned to the opponent during the rollouts.

The second modification adopts a tournament selection [21] approach commonly used in evolutionary algorithms. At each simulation step *k* actions are sampled at random from all available actions. Every selected action is scored according to a heuristic function and action with the highest value is selected as the next move.

Selection and rollout phases rely on a heuristic function to evaluate game states and inform the action and node selection. A heuristic considering selected features of the game was designed and its weights optimized by genetic programming where an agent driven by this

heuristic played series of games against the GreedyOptimizeMove behavior in Metastone with various decks. The heuristic was compared to threat-based heuristic employed by GSV behavior but achieved worse results. For the final experiments the evolved heuristic was replaced by threat-based one.

Basic version of MCTS and MCTS with proposed improvements were matched against artificial players in Metastone. Both approaches achieved win rate near 100% against Random and No-Aggression players. Against GreedyOptimizeMove basic MCTS won 40% of the games while modified MCTS won over 60%. Against the Game State Value player basic MCTS performed with only 21% win rate while enhanced MCTS was able to reach win rate of 42%. Based on the results the proposed MCTS approach performed significantly better than the basic unmodified MCTS.

6.2 MCTS Methods Improved with Machine Learned Heuristic

In [22] an algorithm combining Perfect information MCTS (PIMC) [23] and Information Set MCTS [24] (ISMCTS) is proposed as a solution to the card playing agent in Hearthstone and is further augmented by the use of machine learning.

Transposition table is constructed instead of search tree in order to detect duplicate states reachable with different sequences of actions. To access table entries, information sets, an information observable by active player, serve as keys to the transposition table and value of the entry is a list of available actions executable in the information set where score is assigned to each action. In each iteration a perfect information state is determinized and based on the current information set, available actions in the current state are looked up in the table. If an entry was found, next action is selected according to UCT policy and executed in the current determinization, otherwise the table is expanded with the new entry.

During player's turn the selection of actions which includes playing cards as well as performing attack actions with minions on board can be done in different sequences (permutations). However, when attacking most of the permutations lead to the same resulting state. A heuristic oriented attack solver has been designed to generate an optimal sequence of attack actions for the given state and all attack actions were replaced with a single action called "attack solver" that can be selected at any point during player's turn. This way the MCTS only needs to decide between playing cards, using "attack solver" and ending its turn, reducing the search space of available actions

Lastly, a heuristic function based on a machine learned prediction model is utilized in multiple stages of the MCTS. During the selection stage the heuristic evaluation is included together with the UCB formula via a progressive bias [20]. In simulation phase greedy action selector is utilized to select guide the rollouts with respect to a heuristic. The simulations are limited to a certain depth and the last reached state is evaluated. The evaluation is based on a fully connected neural network which for an input state returns a prediction of the winning player. The neural network is trained from a dataset of vectors describing various game states with an information about the winner created from simulations played between two MCTS bots. The approach reliably wins 100% of games against random oriented player and was matched against Legend rank Hearthstone player (legend is the highest achievable rank in Hearthstone). The human players reported that the artificial player performed well and was challenging to play against.

6.3 Chance Event Bucketing and Machine Learning in MCTS

In this work [25], two improvements to the MCTS are proposed: usage of chance event bucketing and pre-sampling to deal with large branching factor caused by chance nodes and a machine-learned policy to guide rollouts during simulation phase of MCTS.

Perfect Information Monte Carlo Tree Search (PIMC) [23] is utilized to deal with the unknown information, a determinized variation of MCTS that samples number of worlds from the available information and traverses each of the sampled worlds. Results of each world are then summed together and action with the highest score is returned.

To mitigate high branching factor in chance event nodes, chance event bucketing method is applied. This method groups similar possible outcomes of the chance event into buckets and then selects one or more of the outcomes to represent all the possibilities contained within the bucket. Bucketing is applied to the most frequent chance event – card drawing. In Hearthstone, manacost of cards usually reflects their power. This is used as a criterion for bucketing of card draw outcomes, grouping cards with similar manacost into one bucket. Each bucket has different probability based on number of sampled outcomes. When bucketed node is visited during selection, a bucket is selected with respect to the probability and finally one of the sampled outcomes is chosen as the final destination. MCTS with chance bucketing was tested against an AI player available in the silverfish simulator and won 72.3% of games.

To further improve the playing strength of the agent, neural network trained for card playing decisions during the rollout phase of MCTS is employed. The trained policy returns a vector that assigns probability to each card. The probabilities indicate how probable it is for each card to be played in the current state. This trained policy is utilized in the rollout phase for selection of card playing actions. Dependent target selections are then evaluated with a heuristic function and the one with the highest score is selected. MCTS with chance bucketing and the trained policy for rollouts was then tested against the silverfish agent and achieved win rate of 75.3% which is a slight improvement.

6.4 Card Playing Agents Based on Machine Learning

In [26] an approach based only on machine learning techniques is proposed to create 4 different card playing agents. Goal of each agent was to construct an optimal action sequence for the given state.

The first two agents are based on reinforcement learning, rewarding the agent with positive value if victory was achieved and negative value if the game was lost. The search space of both agents is separated into two sections: hand actions consisting of card play and hero power actions, and board actions for selection of attacking minions and their targets. Board models of both agents are learned based on the Q-learning algorithm taken from [27] utilizing multilayer perceptron Q-function in matches against a heuristic-oriented player from [28]. The board model of the first agent selects from up to 57 actions (there can be up to 7 minions on player's half of the battlefield and up to 8 targets for each of them plus an action to do nothing). The second agent separates attacker and target selection from each other and solves both parts individually. The hand model constructs all possible hand action sequences and requests the board model to assign rewards to each sequence. Sequence with the highest score is executed. Both agents utilize the same hand model.

The other two agents are based on an action tree that is built from the available actions. One path from root node to leaf represent an action sequence ending with end turn action. These agents do not separate actions into board actions and hand action and therefore are able to change between them. Traversing the whole action tree would be time consuming, therefore a pruning (possibly alpha-beta) is utilized to quickly find the optimal action sequence. Each agent employs a different metric to measure the leaf node states and guide the pruning algorithm. The first agent utilizes machine learned policy that assigns probability of winning

in state reachable with the action sequence. The second agent is based on machine-learned threat metric, predicting threat level of the player and the opponent and choosing the action that maximizes player's threat and minimizes opponent's threat. Finally an experiment was conducted where each agent played 10 000 games against heuristic oriented agent from [28]. The first Q-learning agent selecting from 57 actions achieved win rate of 32.21%, the second Q-learning based agent solving attacker selection and target selection separately won 44.79% of games played. The third agent utilizing action tree with victory prediction metric reached win rate of 59.55% while the performance of the fourth agent based on a threat level metric peaked at 72.90% win rate.

6.5 Summary

There is a large variety of aspects in collectible card games that can be addressed with the use of the artificial intelligence. To conclude the chapter an overview of the mentioned methods related with creation of artificial players is presented in Table 1:

Section name	MCTS	Determinized MCTS	Expert knowledge	NN	EA	RL	Deck Aprox
MCTS Enhanced with Domain knowledge and Heuristic [19]	✓	✓	✓		✓		✓
MCTS Methods Improved with Machine Learned Heuristics [22]	✓	✓		✓			
Chance Event Bucketing and Machine Learning in MCTS [25]	✓	✓		✓			
Card Playing Agents Based on Machine Learning [25]				✓		✓	

Table 1 - Overview of AI Methods

NN (Neural Networks), EA (Evolutionary Algorithms), RL (Reinforcement Learning)

Each of the presented game playing approaches utilized different simulators and different behaviors for opponents in conducted experiments and therefore it is difficult to predict how these approaches would perform against each other.

7 Approaches

In this section the implementation and description of methods used in the following experiments are discussed.

7.1 Vanilla Monte Carlo Tree Search

Algorithm 1: Vanilla Monte Carlo Tree Search

```
Function: RequestAction(state, player, validActions)
    worldList = sampleWorlds(state, player, worldNum)
    roots = new List<Node>
    for each world in worldList:
        root = new Node (state, validActions)
        currentNode = root
        for number of iterations
            currentNode = currentNode.UCBselect()
            currentNode.expand()
            currentNode.value = rollout(currentNode)
            backpropagate()
        end for
        roots.add(root)
    end for
    actionScores = sumActionScores(roots)
    return getBestAction(actionScores)
```

This is the basic variant of Monte Carlo Tree Search described in section 4.1 designed to play matches in Hearthstone. The approach is depicted in Algorithm 1. To deal with the hidden information represented by the opponent's deck, hand and secrets the artificial player utilized a Perfect Information MCTS approach [23]. In this approach, the hidden information is guessed randomly, constructing a deck with randomly selected cards from the available card pool. This deck is assigned to the opponent and his hand is filled with randomly drawn cards from the assigned deck. If the opponent has secrets attached to their hero they are selected randomly as well. The random determinization only selects from cards that are available to the hero the opponent plays with, therefore it cannot give a Priest card to an opponent who plays as a Warlock (although in real game of Hearthstone Warlock might get access to some of the Priest specific cards during gameplay due to effects of some cards). Before the MCTS search loop, number of possible randomized worlds are sampled beforehand and each of them is searched individually. Scores of each available action is then summed together across all the worlds and the action with the highest score is returned. To balance between exploration and exploitation during selection phase the Upper Confidence Bound is utilized. Random driven rollouts are performed up to a terminal state of the game to assign scores to nodes in the search tree.

7.2 Enhanced Monte Carlo Tree Search

This approach utilizes enhancements to Monte Carlo Tree Search proposed in [19] and discussed in section 5.1. The implementation of the approach can be found [14]. The enhancements are taken from the available implementation and are fitted into the Vanilla MCTS.

Algorithm 2: Deck Determinization

```
Function: AssignDeckToOpponent(opponent)
    playedCards = opponent.getPlayedCards()
    decks = deckDatabase.getDecks()
    for each deck in decks
        deck.co-occurrences = cardsInDeck(playedCards, deck)
    opponent.deck = getDeckWithMostCo-occurences(decks)
    opponent.deck.removePlayedCards(playedCards)
    handSize = opponent.hand.size()
    opponent.hand = drawNCards(handSize)
end procedure
```

During the selection phase of MCTS algorithm a domain specific knowledge is added to the UCB selection formula. This knowledge is given by the Threat Based heuristic utilized by GSV behavior which is used to evaluate game state associated with the node. This knowledge has high impact on node selection within first few iterations when the construction of the tree has just begun, reflecting the decision-making of GSV behavior.

To deal with the hidden information usage of deck database is proposed. Hearthstone players typically keep track of momentarily popular decks for each of the heroes. During gameplay they assume their opponent uses one of the momentarily popular decks based on the hero selection of the opponent. With each card played by the opponent the player receives additional information that can be used to further reduce the number of possibilities. The usage of deck database is inspired by this thought process. The bot utilizes a set of preconstructed decks that is assigned to the opponent during rollouts based on the cards played by the opponent. The deck with the most cards in common with the played cards is assigned. Cards already played by the opponent are removed from the deck and cards in hand are drawn randomly from the assigned deck. This is done before the execution of rollout function. Pseudocode is displayed in Algorithm 2. In practice, this is an effective way to cope with unknown contents of opponent's deck because players tend to use the decks that perform the best in the current environment. Although there are usually numerous variations of popular decks, the difference is often in only few cards. Of course, if the opponent plays with a deck that is completely different from decks in the database, the artificial player would determine the unknown information completely wrong which would have negative effect on

its performance. However, even an experienced human player is unable to guess the opponent's deck in such a situation.

Algorithm 3: Rollout Action Selection

```
Function: RolloutBehavior(state, player, opponent)
    validActions = state.getValidActions()
    sampledActions = tournamentSelection(validActions)
    for each action in sampled actions
        action.score = heuristic.evaluate(action)
    end for
    return greedilySelectAction(actions, player, opponent)

Function: tournamentSelection(validActions)
    n = validActions.size * samplingPercentage
    sampledActions = randomlySelectnActions(validActions, n)
    return sampledActions

Function: greedilySelectAction(actions, player, opponent)
    if activePlayer is player
        return actionWithHighestScore(actions)
    if activePlayer is opponent
        return actionWithLowestScore(actions)
```

To increase the credibility of simulations during the rollout phase the approach makes use of heuristic evaluation to guide the action selection during rollouts. In the work [19] a heuristic was designed that considers selected components of the game state to evaluate it. The weights were then tuned by application of evolutionary algorithms and compared to Threat Based heuristic of GSV behavior. The threat-based heuristic proved to achieve better performance as its weights were also tuned by evolutionary algorithms but considers more components than the proposed heuristic. In the end, the threat-based heuristic was used in the final configuration of the approach and therefore the evolutionary approach is not examined in this work. To guide the action selection during rollouts each player is assigned with a greedy action selector that utilizes threat-based heuristic for state evaluation. However, with high number of iterations the rollouts can be very time-consuming. To alleviate this issue a tournament selection is applied whenever an action is required. Tournament selection reduces the amount of valid actions be sampling a percentage of the available actions at random and the action selector considers only the sampled actions. Sampled actions are then evaluated by the heuristic function. The action selector distinguishes between the players, selecting action with the highest score if artificial player is the active player and action with the worst score if active player is the opponent. The functions utilized to guide the rollout decision-making are viewed in Algorithm 3.

Finally, during the player's turn the approach attempts to reuse the tree constructed in previous search. When an action is return, the node and state associated with it is saved. When artificial agent is requested to select an action, it compares the current game state with the remembered state and if these states are marked identical, the saved node is selected as current root. Otherwise, new root is initialized and the tree construction begins anew. To

determine whether the states are identical, valid actions of the current state are compared with actions that can be performed in the saved state. If all the actions associated with the saved node are among the valid actions of the current state, the states are considered identical.

7.3 MCTS with Chance Event Bucketing and Neural Network

This artificial agent is inspired by techniques described in section 5.3 and the methods are incorporated to the Vanilla MCTS from section 6.1.

The approach employs depth-limited simulations during rollout phase. Whenever a simulation is launched it is played until either a terminal state or the limit of played turns is reached. The simulation counts number of executed end turn actions and when the counter is higher than the depth the simulation is terminated. If reached state is not terminal it is evaluated with a heuristic function. However, in the mentioned work heuristic available in the Silverfish simulator was used for this purpose. Because this heuristic is not present in the Metastone simulator the threat-based heuristic is utilized instead. The reached state is evaluated from player's and opponent's perspective and the one with higher heuristic value is predicted as the winning player (the higher the heuristic value the higher threat the player represents towards his opponent).

To incorporate randomness into the tree search process, chance event bucketing method is applied. Specifically, this method is applied to the most common chance event that occurs throughout the game, the card drawing. Chance event bucketing takes all possible outcomes of a chance event and groups them into number of buckets based on a selected criterion. Then, for each bucket some of the outcomes are selected as representative values. In this case, the bucketing criterion is the manacost of cards. Figure 5 depicts an example where a card draw event is separated into 4 buckets - 1 containing cards with manacosts 0-2, 1 with cards with manacosts 3-4, 1 for cards with manacosts 5-6 and 1 for cards with manacost 7 and above - and each bucket has 2 representative values. When transitioning to bucketed node during selection phase of MCTS, one of the buckets is selected as a destination with respect to probability that is defined by the number of samples contained within the bucket compared to the total number of samples. In Figure 5 each bucket has the same probability, but if the first bucket contained 4 samples and total number of samples was 10 the probability of transitioning to the first bucket would be 40% and probability of other

buckets would be 20%. Finally, one of the samples within the bucket is selected at random as the final outcome. This way effects of random events are introduced into the scores of tree nodes that are associated with the chance events. To give even better estimation of the node's score, more than one bucket could be explored when transitioning to a chance node. The approach in this thesis distributes outcomes of card draw events into 5 buckets with manacosts 0-1, 2, 3-4, 5-6, 7+. If 1 card is drawn 10 possible outcomes are sampled in total among all the buckets. At least one sample is assigned to a bucket if there is at least 1 card that fits the bucket's criterion. The remaining samples are then distributed among the buckets with respect to the number of their possible outcomes (bucket with the most possibilities contains the highest number of samples). If 2 or more cards are drawn 5, possible outcomes are sampled for each card draw instead of 10 (if 2 cards are drawn there are 5^2 possible outcomes in total).

Algorithm 4: Chance Event Bucketing

```
Function: Select(root)
   current = node
   while current is fully expanded
      current = UCTselect(current)
      while current.buckets is not null
         current = current.buckets.getOutcome()
   return current

Function: expand(nodeToExpand)
   action = unexpandedTransitions.getRandomAction()
   node = new Node()
   if action draws a card
      node.buckets = sampleOutcomesIntoBuckets(action)
   else
      node.state = nodeToExpand.state.performAction(action)
   return node
```

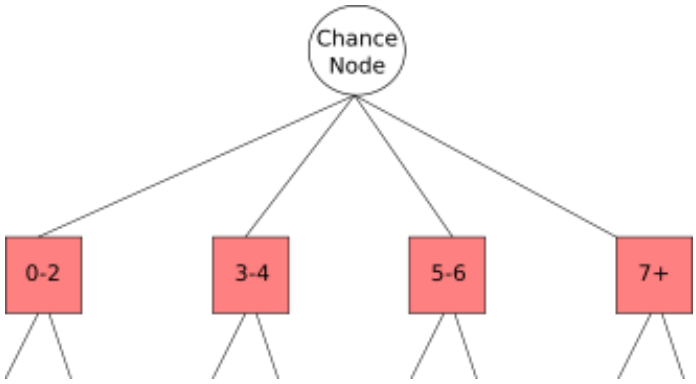


Figure 5 – Example of chance node bucketing

To further improve performance of the approach, machine-learned card play policy utilizing neural network is employed to guide the action selection during rollouts. The goal of the neural network is to assign probability to each card that describes how probable it is for a card to be played in the current situation. However, some cards may require dependent action after being played such as target selection. Once a card play action has been selected its dependent actions are resolved using greedy heuristic-oriented action selector. Therefore,

whenever an action needs to be selected the current state is encoded and evaluated by the neural network. Network outputs probability for each card and card in hand with the highest probability is played. If no card play actions are available among the valid actions the remaining actions are resolved by the action selector. As the action selector, GreedyOptimizeMove behavior available in the Metastone simulator is used.

To encode the state of Hearthstone for the neural network, 3 features are considered:

- **Global features**: Global features are represented as a single vector encoding current health of each player using a 5-bit binary representation for each player, the player's remaining mana, the opponent's available mana on the next turn, whether the player and opponent have a weapon equipped, which of the two players played first and whether the total attack value of player's minions is higher than the total health of opponent's minions.
- **Hand features**: A vector that one-hot encodes contents of each players hand. Each card in the simulator is encoded with a vector that represents how many instances of the card are in each player's hand, whether the card is playable by the players and if there is a possible follow-up card play after the card is played.
- **Board features**: A single vector one-hot encoding the current state of the game board. Instances of each minion that can appear in the game are encoded into a matrix where rows represent current health value (5 different health values are distinguished – 0-1, 2-3, 4-5, 6-7, 8+) of the minion and columns stand for the number of instances of the minion with the particular amount of health controlled by the players. This matrix is then flattened into a single vector and an information whether the minion is a legendary minion and whether it has an aura (an effect that does something while the minion is on board) is appended.

The training data for the neural network was generated from simulations between two GameStateValue behaviors, which is the strongest artificial player offered by the Metastone simulator. 27 000 open-handed mirror matches were played with each of the 3 decks that were selected for the following experiments (Pirate Warrior, Midrange Shaman, Freeze Mage) for a total of 81 000 matches. Every turn with different maximum mana value from each player's perspective (for a total of 20 turns) is observed and all card play actions performed in these turns are encoded and saved into the training file with the played card as the expect output of the encoding. In the end the training data file contained about 4 million training samples. It is important to note that the size of

encoding depends on the number of cards available in the simulator. Metastone simulator offers nearly 1 300 different cards and encoding of a single state included about 60 000 values. File that contained training samples from only 1 000 matches required about 3.5GB of memory space. To save data from 81 000 matches would require over 280GB. To lower the memory requirements the available card pool was limited to 100 cards consisting of each card that appears in the 3 selected decks and a few additional cards to reduce accuracy of random determinization of hidden information used by Vanilla MCTS and this approach.

The neural network used for card play action selection employs feedforward neural network topology consisting of 3 hidden fully-connected layers (Figure 6). Each of the layers utilizes leaky ReLU activation function with parameter $\alpha = 0.2$ and 50% dropout to handle overfitting. The output layer outputs k values where k is the number of different available cards. For parameter initialization Uniform Xavier Initialization was used. For training the adaptive moment estimation (ADAM) with parameters $\alpha = 10^3$, decay $\sqrt{t/3}$, $\beta_1 = 0.9$ and $\beta_2 = 0.999$ is used with minibatch size of 200. The network was trained for 5 epochs, learning for about 23 hours.

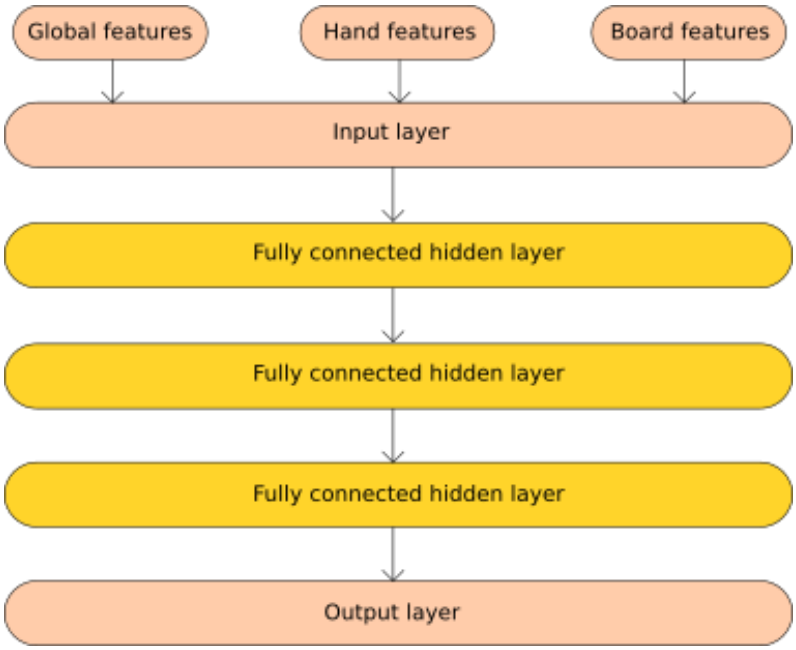


Figure 6 – Feed-forward network architecture

7.4 Summary

In this chapter the approaches used in the following experiments and their implementation are discussed. First, the basic Monte Carlo Tree Search in a form of Vanilla MCTS and its application to a game of Hearthstone is examined. Then the Vanilla MCTS was enhanced with methods mentioned in section 5.1, creating first enhanced version of the MCTS algorithm, and finally a second improved version of MCTS was developed with the use of chance event bucketing and rollouts guided by a machine learned policy from section 5.3.

8 Experiments

In this chapter the experiments for the implemented approaches are conducted. In these experiments the proposed approaches are tested against behaviors that are available in the Metastone simulator, namely PlayRandomBehavior (Random) that chooses random actions, OptimizeMoveBehavior (Greedy) which evaluates each available action with heuristic function and chooses the one with the highest score, and GameStateValueBehavior (GSV) utilizing Alpha-Beta pruning algorithm driven by Threat Based heuristic and is the strongest of the Metastone behaviors. For the experiments the limited card pool mentioned in section 6.3 is used and 3 different decks are selected where each represents one of the general strategies (Aggressive, Midrange, Control):

- **Pirate Warrior**: Aggressive deck with goal to kill the opponent's hero as fast as possible
- **Midrange Shaman**: This deck concentrates on maintaining dominance over the battlefield with usage of stronger (but more expensive) minions and spells
- **Freeze Mage**: Deck based around control strategy. The goal of the deck is to eliminate all threats from the board with the usage of spells and to survive for as long as possible until its win condition is drawn (typically a combination of cards that is able to kill the opponent's hero in 1 or 2 turns)

In each individual experiment 5 series of 50 simulations for a total of 250 matches are played between 2 selected bots where both players have the same deck (mirror matches). Each bot has unlimited thinking time to make its decision.

8.1 MetaCentrum

MetaCentrum [30] [31] is an activity of CESNET association that operates and manages National Grid Infrastructure (NGI) in Czech Republic as part of the pan-European infrastructure built in the framework of the EGI [32] project. MetaCentrum supports research projects in many research disciplines and enables researchers to easily share computing and storage resources. Registered users have access to available resources and are allowed to use them free of charge. Grid infrastructure managed by MetaCentrum was utilized to conduct experiments in this chapter.

8.2 Vanilla MCTS

Vanilla MCTS from section 6.1 is the basic implementation of PIMC that utilizes random oriented rollouts to estimate value of available actions and deals with hidden information by randomly sampling the unknown data and creating number of possible worlds.

8.2.1 Parameter Selection

Performance of Vanilla MCTS depends on 3 parameters: Number of iterations, number of sampled worlds and exploration parameter C used in UCT. To test various values for the parameters a round-robin tournament with Pirate Warrior deck is run with 250 games per match and afterwards each candidate is matched against GSV bot. Pirate Warrior is selected because in mirror match it is important to alter between aggression and board control based on current state of the game.

8.2.1.1 Exploration parameter

First the effect of the exploration parameter on Vanilla MCTS is explored. The candidates for the exploration parameter are {0.5, 0.6, 0.7, 0.9, 1.1}. Number of iterations is set to 250 and number of sampled worlds to 7.

According to the results in Table 2 the performance of parameters 0.5, 0.6 and 0.7 is similar and with further increase of the parameter the performance begins to decline. With too large exploration parameter the algorithm will not spend as much time exploiting the promising moves and will expand the search tree more evenly which could lead to selection of worse moves. From now on value 0.7 will be used for the Vanilla MCTS as it has the most balanced results and seemingly the best performance against GSV player.

	0.5	**0.6**	**0.7**	**0.9**	**1.1**	**GSV**
0.5	-	47.6	49.6	54.4	55.6	36.8
0.6	52.4	-	45.2	50.8	57.6	35.6
0.7	50.4	54.8	-	51.6	50.4	38.0
0.9	45.6	49.2	48.4	-	47.6	29.6
1.1	44.4	42.4	49.6	52.4	-	31.6

Table 2 – Exploration parameter results

8.2.1.2 Iterations

The number of iterations defines how many expansions occur during the algorithm's execution and how many simulations are played out to estimate the value of each node in the tree. The values are selected from set of parameters {100, 250, 500, 750, 1000} and for each value its performance and time consumption are measured. Exploration constant is set to 0.7 and number of sampled worlds to 7.

In table 3 are summarized experiment results. Surprisingly in round-robin tournament it seems that number of iterations has no significant effect on the performance of MCTS, however when matched against GSV low number of iterations show low performance compared to higher values as is expected because more iterations mean more thinking time until deciding. Performance peaks at 500 iterations and further increases seem to have no significant effect on the playing strength.

	100	**250**	**500**	**750**	**1000**	**GSV**
100	-	50.0%	49.6%	46.4%	50.4%	22.4%
250	50.0%	-	45.2%	46.4%	50.0%	28.4%
500	50.4%	54.8%	-	48.8%	49.6%	32.4%
750	53.6%	53.6%	51.2%	-	50.0%	31.2%
1000	49.6%	50.0%	50.4%	50.0%	-	30.8%

Table 3 – Iterations results

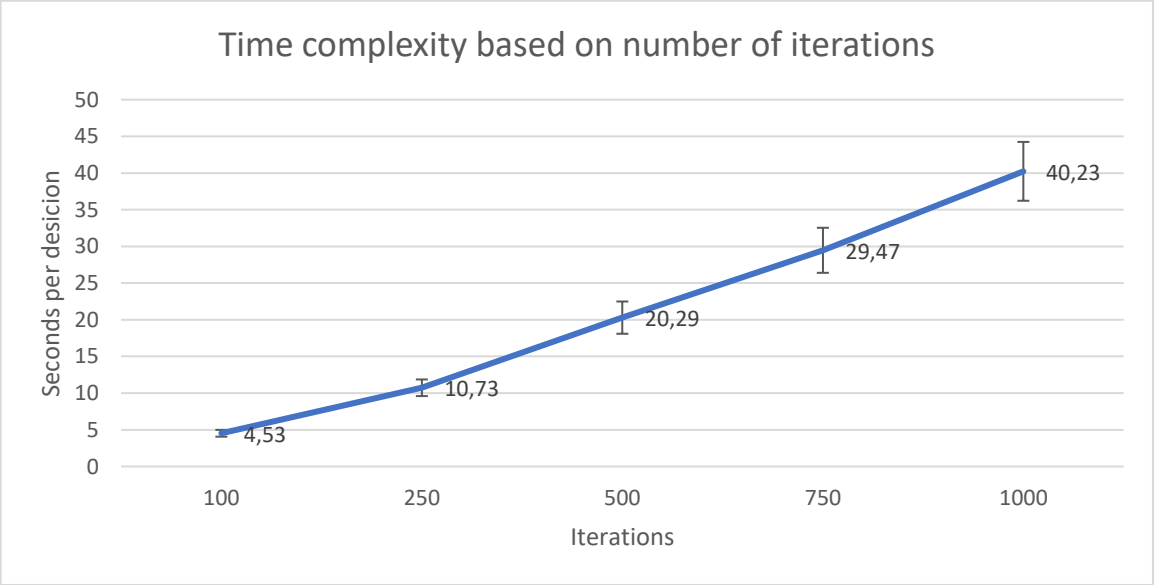


Figure 7 – Time complexity base on iterations

In Figure 7 the average thinking time for action selection in seconds is overviewed with vertical lines representing 95% confidence interval for each selected iteration. As was anticipated, increasing the number of iterations increases the time needed to decide. For the following experiments value of 500 iterations will be used as it is sufficient enough to achieve reasonable results. However, it is important to note that in game of Hearthstone each player has 75 seconds to play their turn and can perform multiple actions until they decide to end it. With 20 seconds per action selection the player would be able to perform only 3 actions until their turn is ended by the game which might be only a small fraction of all the actions that could have been done. Reducing the amount of time needed to make a decision or distributing available time budget to available actions could be explored in further works.

8.2.1.3 Worlds

This parameter determines the number of randomly sampled worlds based on the current state to deal with the hidden information (opponent's hand, deck and secrets). The set of candidate values for this parameter is {5, 7, 9, 11, 13}. Performance and time requirements of each value is examined. Number of iterations is set to 500 and exploration parameter to 0.7.

The results in table 4 indicate that increasing the number of sampled worlds increases the performance of the Vanilla MCTS. With few samples the search may not encounter some of the possible moves the opponent could perform and therefore the MCTS could not consider these moves when selecting its next action. Increasing the number of samples allows the player to search higher number of different possible situations and chose an action that performs well in all the considered situations.

	5	**7**	**9**	**11**	**13**	**GSV**
5	-	47.6%	42.8%	40.4%	42.8%	27.2%
7	52.4%	-	52.4%	43.2%	40.0%	35.2%
9	57.2%	47.6%	-	49.2%	44.8%	36.8%
11	59.6%	56.8%	50.8%	-	48.4%	37.6%
13	57.2%	60.0%	55.2%	51.6%	-	38.4%

Table 4 – World results

Figure 8 shows the average time in seconds needed to decide based on the number of sampled worlds. Similarly to iterations, increasing the number of worlds increases the time complexity because each of the sampled worlds needs to be explored. 11 worlds are chosen for the next experiments as a setting that presents sufficient playing strength while requiring reasonable amount of time to decide.

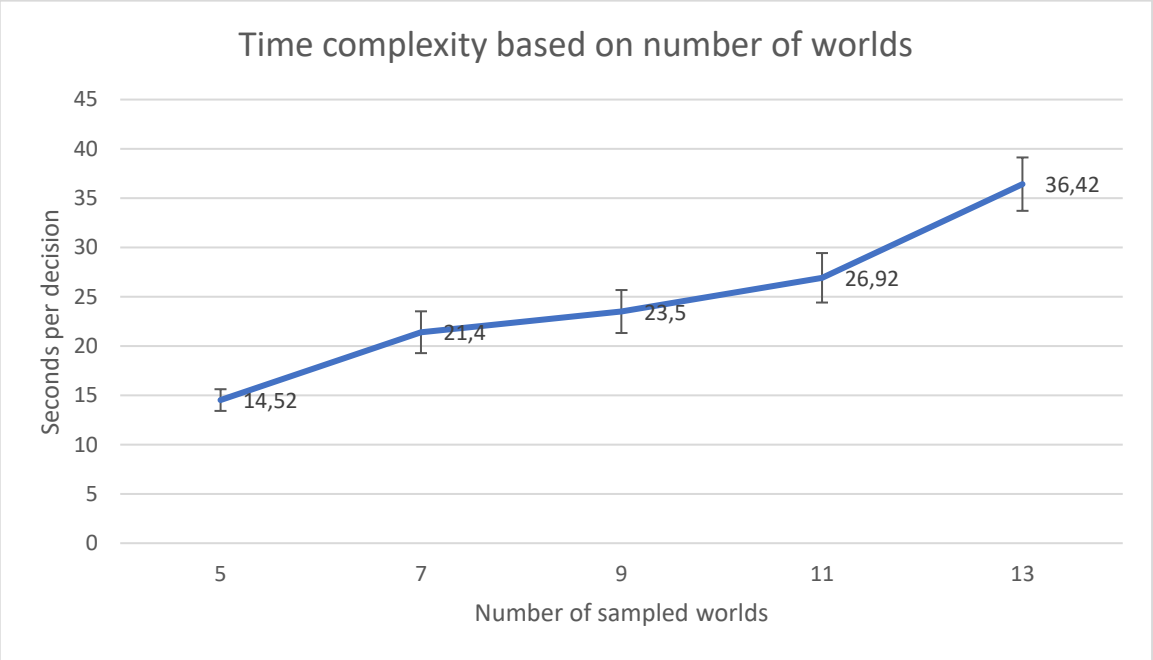


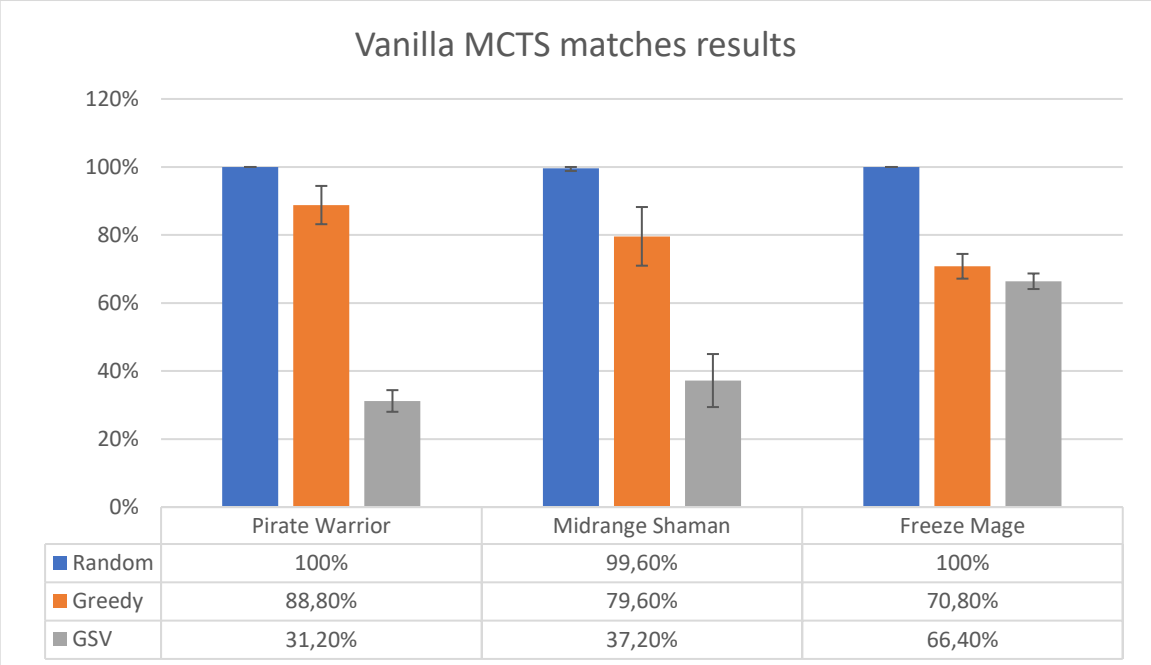
Figure 8 – Time complexity based on worlds

The final configuration of Vanilla MCTS is:

- 0.7 exploration parameter
- 500 iterations
- 11 sampled worlds

8.2.2 Matches against Metastone Behaviors

To conclude the section about Vanilla MCTS the approach is matched against the artificial players in the Metastone simulator with results depicted in Figure 9. Vanilla MCTS with the selected parameters is able to win every game against randomly playing agent and wins majority of matches against the Greedy behavior, achieving win rate of 70.8% in Freeze Mage mirror match, 79.6% with Midrange Shaman and 88.8% against Pirate Warrior. Against GSV behavior Vanilla MCTS presented an underwhelming performance in Pirate Warrior and Midrange Shaman mirror matches with 31.2% and 37.2% win rates respectively. However, it shows substantially better performance in the Freeze Mage mirror match achieving win rate of 66.40%. This significant win rate difference between Freeze Mage and other decks is likely caused by the need of resource management and planning during the gameplay with the Freeze Mage deck. In order to win the game the player needs to preserve the cards that are part of the deck's winning card combination. If these cards are used early in order to control the board or without the presence of other combo pieces the players might put themselves further from the winning position or even make it near impossible for them to win. The GSV does not plan past its own turn and therefore is unable to realize this, resulting in worse performance.



	Pirate Warrior	Midrange Shaman	Freeze Mage
Random	100%	99,60%	100%
Greedy	88,80%	79,60%	70,80%
GSV	31,20%	37,20%	66,40%

Figure 9 – Results of games between Vanilla MCTS and Metastone behaviors

8.3 MCTS Enhanced with Domain Knowledge and Heuristics

In this part experiments for the approach proposed in sections 5.1 and 6.2 that utilizes heuristic oriented rollouts, deck database to deal with the unknown information and random action sampling are performed. For the experiments the most successful parameter configuration carried out in the work [19] is selected. However, this configuration includes usage of tree reuse during players turn. Because of the random nature of Hearthstone performing the selected action might lead to a different state than the expect state saved in the search tree. Such a state has not been explored and the search tree needs to be constructed anew. In the implementation each random event is represented with 1 possible outcome. It is improbable that the sampled outcome is also contained within the search tree and any time a chance event is encountered the tree cannot be reused. To determine whether the usage of tree reuse has effect on the performance of the artificial player an experiment is conducted. 6 series of 50 simulations for a total of 300 matches are played between one bot utilizing tree-reuse and the other without the tree-reuse method. Pirate Warrior deck is selected for the experiment as it has the lowest quantity of cards with effects causing random events from the selected decks. Table 5 contains results of the experiments. Win rate of both approaches is close to 50%. The 95% confidence interval of the bot utilizing Tree-reuse is $< 43.59;\ 52.41 >$, therefore the usage of the Tree-reuse method has no significant effect on the playing strength of the artificial player.

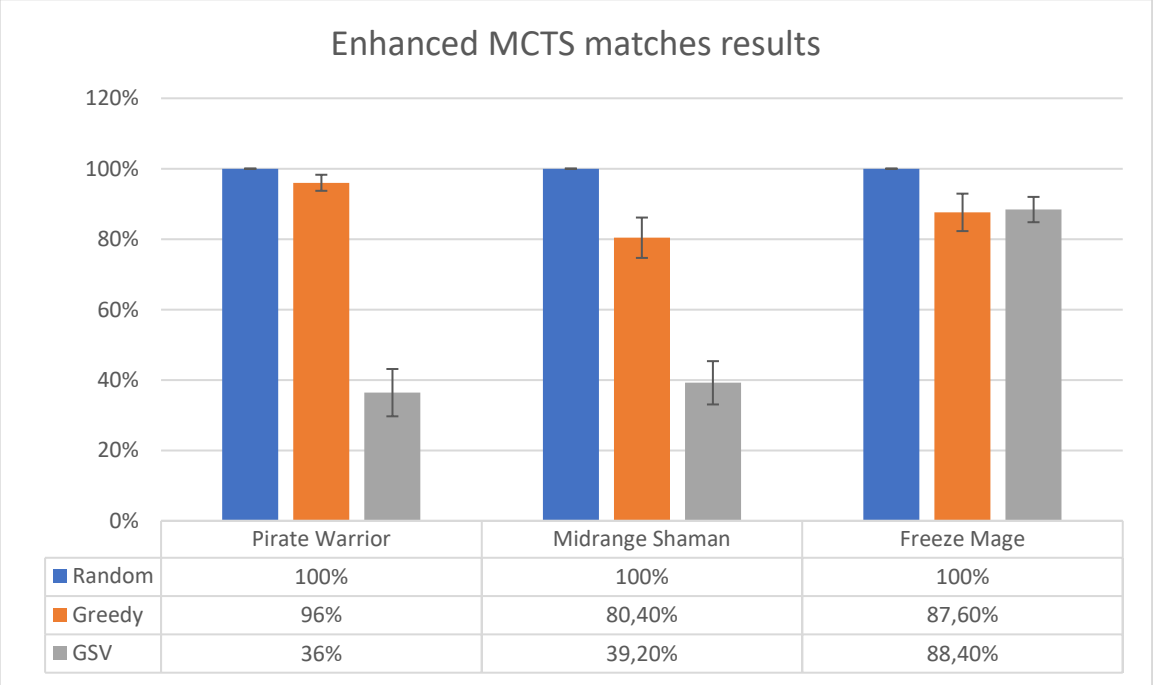
	Win rate	**Standard Deviation**	**95% Confidence Interval**
Tree-reuse	48%	5.513	$< 43.59;\ 52.41 >$
No Tree-reuse	52%	5.513	$< 47.59;\ 56.41 >$

Table 5 – Tree-reuse vs no Tree-reuse

The final selection of parameters for this approach looks like this:

- Iterations: 60 (number of tree expansions)
- Rollouts: 20 (number of rollouts executed to determine score for the expanded node)
- Action sampling for the player: 75%
- Action sampling for the opponent: 50%
- Heuristic: Threat Based Heuristic (heuristic used by GSV)
- Tree-Reuse: No

Now the approach is matched against the available behaviors in the Metastone simulator. The average win rates and 95% confidence intervals of these matches are depicted in Figure 10. The enhanced version of MCTS was able to win every game against the random playing behavior, reaching 100% win rate in each mirror match similarly to the Vanilla MCTS. It is also able to reliably defeat the Greedy behavior driven only by heuristic evaluation with win rate of 96% with Pirate Warrior, 87,6% with Freeze Mage and 80,4% in Midrange Shaman mirror match which is an improvement compared to the Vanilla version. When matched against the GSV behavior the bot is underperforming in Pirate Warrior match with 36% of the games won and against Midrange Shaman with 39.20%. These results are very similar to those achieved in the work that inspired this method. In Freeze Mage mirror match the enhanced approach shows excellent performance with win rate of 88,4%, winning most of the matches. This is a significant improvement compared to the Vanilla version that is likely caused by the ability to accurately determine the contents of the opponent's deck. It is worthwhile to note that this approach with its current parameter setup needs 11.67 seconds on average to decide. The average is 2.3 times lower than the average time complexity of Vanilla MCTS, however its performance is higher. Win the same thinking time as Vanilla MCTS it would likely be able to achieve even better results than with the current configuration with only 60 iterations.



	Pirate Warrior	Midrange Shaman	Freeze Mage
Random	100%	100%	100%
Greedy	96%	80,40%	87,60%
GSV	36%	39,20%	88,40%

Figure 10 – Results of games between Enhanced MCTS and Metastone behaviors

8.4 Chance Event Bucketing and Neural Network

The experiments regarding addition of depth limited search, chance event bucketing and neural network to guide rollouts are carried out in this section. First, various selections for depth parameter are explored, then the effects of chance event bucketing and depth limited search on the performance are examined and finally neural network is applied to the rollout phase of the approach and matched against Metastone behaviors.

8.4.1 Depth parameter

The approach utilizes depth limited rollouts to estimate value of tree nodes. Rollout is played until certain depth and if reached state is not terminal it is evaluated with threat-based heuristic used by GSV and player with higher heuristic value is predicted as a winner. The candidate set of values for the experiment is {5, 7, 9, 11, 13}. The tests are run with Pirate Warrior deck, 500 iterations, 11 worlds and 0.7 exploration parameter against Metastone behaviors. Round Robin tournament is not performed because addition of depth limit caused issue where the bot utilizing the method labeled as player 1 lost almost every match in the tournament regardless of the selected parameters. (being player 1 does not necessarily mean the player goes first. Number 1 resembles identification number to distinguish players in the simulator). Based on observation of simulations the bot made reasonable decisions and the reason for this unexpected behavior is unknown. Results of candidate depths are contained in table 6. According to the data, artificial player reaches peak performance with 7 depths and then begins to decrease with increasing depth value. The decrease of performance with higher depth is surprising. The reason for this could be that with Pirate Warrior the player is able to empty their hand in the first 4-5 turns and available actions in future turns heavily depend on cards drawn. Using only 1 sample for card draw outcomes does not provide sufficient score estimation of available actions.

	5	**7**	**9**	**11**	**13**
Random	100%	100%	100%	100%	100%
Greedy	96.4%	97.2%	96.0%	93,6%	92,0%
GSV	56.6%	57.2%	52.8%	47.6%	47.6%

Table 6 – Depth results

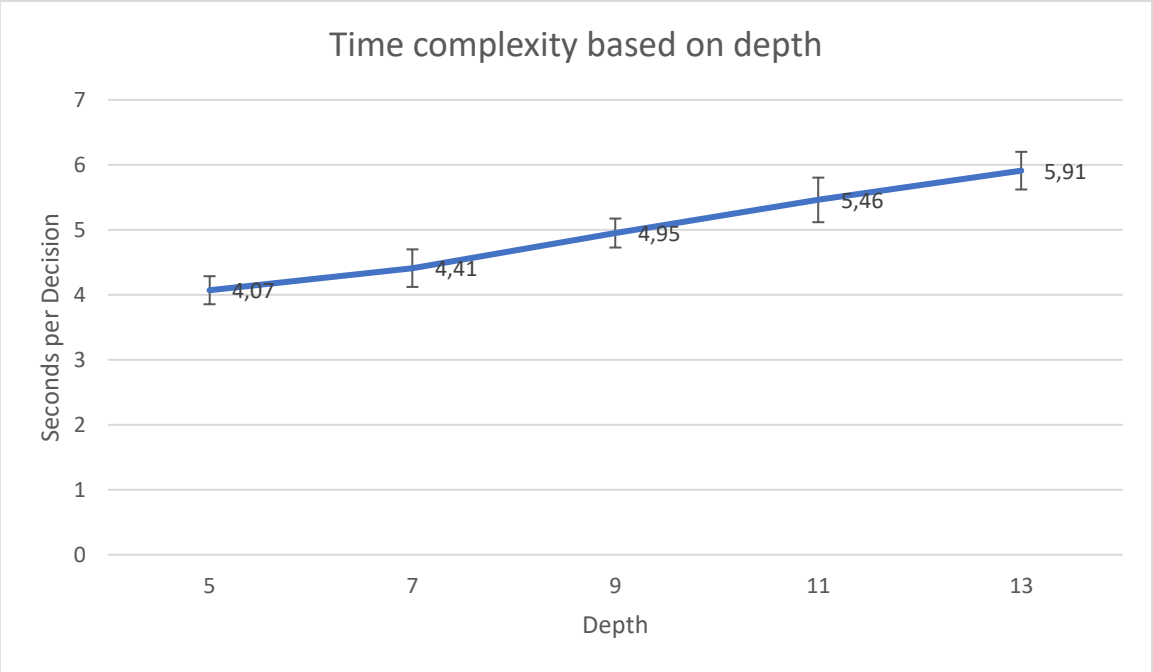
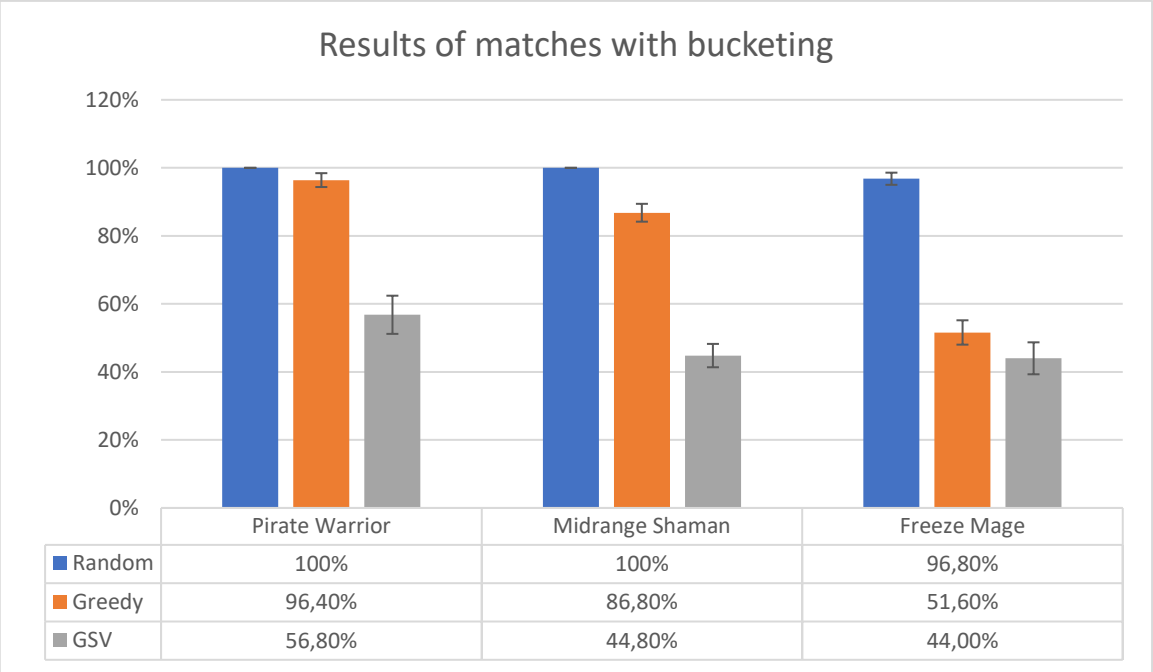


Figure 11 – Time complexity based on depth

In Figure 11, time in seconds needed to decide based on selected depth value is depicted. The raise of depth increases the time complexity which is anticipated. The approach employs the same parameters as Vanilla MCTS, however it needs nearly 7 times less decision time with 7 depths while achieving better performance. This is a significant improvement because with 4.41 seconds to decide the approach could be utilized as artificial player in a real game of Hearthstone.

8.4.2 Chance Event Bucketing

Chance event bucketing separates possible outcomes into number of buckets where each contains similar outcomes. Results in Figure 12 show that this approach performs significantly better in Pirate Warrior and Midrange Shaman mirror matches against GSV behavior with win rates of 56.8% and 44.8% respectively in comparison with Vanilla MCTS, but on the other hand achieves worse results in Freeze Mage mirror match with 96.8% win rate against Random behavior, 51.6% against Greedy behavior and 44.0% against GSV which represents a decrease in comparison with Vanilla MCTS. However, the results of the Pirate Warrior matches are very similar to values in Table 6. The differences between win rates are likely caused by the addition of depth limited rollouts and not the bucketing. To support this claim more experiments are conducted without the use of depth limited search.

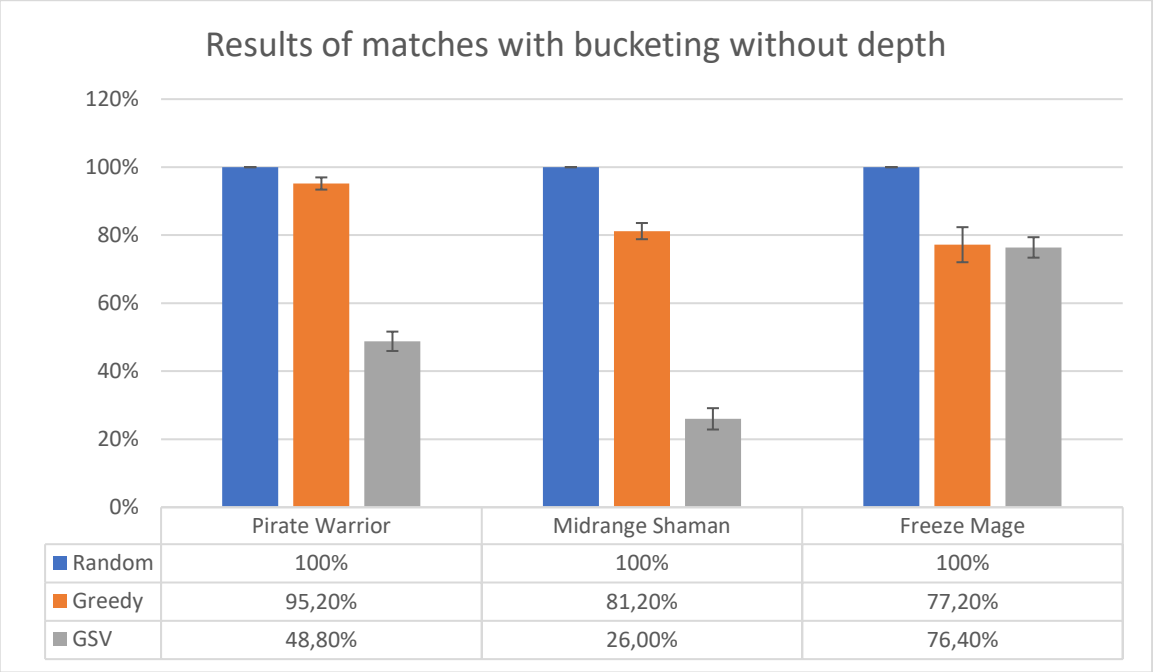


	Pirate Warrior	Midrange Shaman	Freeze Mage
Random	100%	100%	96,80%
Greedy	96,40%	86,80%	51,60%
GSV	56,80%	44,80%	44,00%

Figure 12 – Results of MCTS with card draw bucketing

According to results in Figure 13 the radical change in win rates is indeed caused by the use of depth limited search, but only partially. Without the depth limit the artificial player performed significantly better in the Freeze Mage mirror matches, winning 76.4% of matches against GSV, as it was able to better plan the usage of its resources with unlimited depth which is crucial in Freeze Mage deck as was mentioned earlier. In Pirate Warrior and Midrange Shaman matches the bot performed worse than in the previous experiment. However, in comparison with the Vanilla MCTS the addition of card draw bucketing had beneficial effect on Pirate Warrior (from 31.2% to 48.8%) and Freeze Mage (66.4% to 76.4%) mirror matches against both GSV and Greedy behaviors. This supports the previous claim about decrease of win rate of Pirate Warrior deck with increasing depth caused by high dependency of available actions based on cards drawn. With only one possible outcome present in the tree the action selection is centered around the single selected outcome, however with bucketing the action is selected with respect to other outcomes as well, providing better score estimation for available actions. Therefore, the usage of bucketing alleviates deprecation of score estimation of actions with increasing depth. However, in Midrange Shaman mirror match its performance against GSV rapidly decreased. This decrease is surprising because the addition of bucketing was expected to help in matches where card drawing events happen frequently. In Pirate Warrior deck cards are drawn only at the beginning of each turn. On the other hand Freeze Mage deck contains high number of

cards that draw additional cards and the event can happen multiple times during player's turn. In both cases, bucketing had positive effect on player's performance. Midrange Shaman deck has only few cards that invoke card draw event. Card drawing happens more frequently than in Pirate Warrior deck but with much lower frequency than in Freeze Mage deck, but addition of bucketing reduced its performance against GSV while retaining similar performance against Greedy behavior.



	Pirate Warrior	Midrange Shaman	Freeze Mage
Random	100%	100%	100%
Greedy	95,20%	81,20%	77,20%
GSV	48,80%	26,00%	76,40%

Figure 13 – Results of bucketing without depth limited search

8.4.3 Neural Network

In this part the effects of neural network application to simulations of rollout phase is examined. The neural network was trained to mimic the card play action selection of GSV behavior. First, the accuracy of neural network's predictions is tested. To measure the accuracy, 100 mirror matches with each deck are played between GSV players and each card play action of players is compared to the prediction of the neural network. Network achieved only 34.22% accuracy with Pirate Warrior, 35.22% with Midrange Shaman and 28,40% accuracy in Freeze Mage mirror match, reaching only 32.61% average accuracy. On top of that, one evaluation of the neural network takes 43.85 milliseconds. With the selected parameters for the MCTS in previous sections 5 500 simulations are played up to a depth of 7, playing at most 44 000 turns until a decision is made. If 1 card play action was executed every turn 44 000 evaluations via the neural network would be required. This process would

take 1 929 400 milliseconds which is about 32 minutes required to make a single card play decision. In comparison with the outputs in [25] where the deep feedforward neural network supposedly achieved 74.725% accuracy on average requiring only 140 microseconds for an evaluation the results of the implemented neural network are very discouraging. The architecture and configuration of the neural network is design according to the description in the mention work, however the differences are significant. The difference in accuracy could possibly be caused by different amount of time used to learn the network, however the cause of the radical difference in evaluation's time complexity is unknown, especially after considering that the game state encoding in this thesis is smaller than the encoding in the mentioned work thanks to the usage of limited card pool.

In order to perform experiments with the neural network the time required to decide is reduced by decreasing parameters of Monte Carlo Tree Search to 100 iterations, 6 sampled worlds and depth 6. With this configuration, selecting a card play action requires about 3 minutes. Even with the reduced configuration simulating 1 game might take several hours, therefore the experiments are reduced to 5 series of 20 simulations for a total of 100 games per mirror match. Matches against random playing behavior are not performed because the win rate against the bot would most likely be 100% as it was in every previous experiments. According to data in Figure 14, the approach performed significantly worse in the Pirate Warrior and Midrange Shaman mirror matches. This decrease is cause by the reduction of the MCTS parameters. Therefore, the tree is not searched as thoroughly as it was in previous experiments. On the other hand in the Freeze Mage mirror match the approach utilizing the neural network achieved average win rate of 79% against the GSV behavior which is the second highest win rate against Freeze Mage among all the experiments conducted even with the reduced parameters. Apparently, the replacement of randomly oriented rollout by a reasonable action selector in the MCTS utilizing the depth limited search and card draw event bucketing had very beneficial effect on the performance in the Freeze Mage mirror match, increasing win rate from 44% to 79%. If faster and more accurate action selector to guide the simulations during the rollout phase was utilized while retaining the original parameter configuration this approach could prove as a strong artificial player with reasonable amount of time to decide.

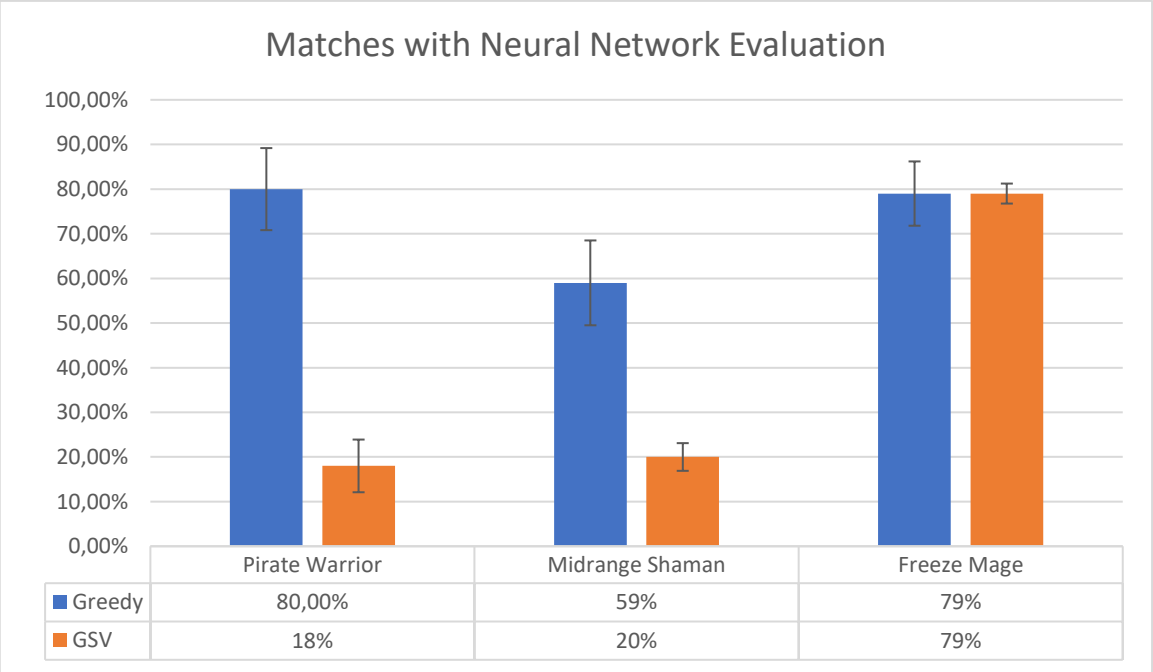


Figure 14 – Results of matches with the usage of the learned card play policy

8.5 Comparison of Methods

In this final section the implemented approaches are matched against each other to compare their playing strength. Because of reduction of parameters of the MCTS utilizing neural network which ultimately reduced performance of the depth limit and card draw event bucketing methods, both these MCTS variations are tested individually against other agents. 5 series of 50 simulations for each mirror match are played except for the matches where neural network is present in which the number of simulations is reduced to 20 per series. Figure 15 depicts the win rates of MCTS utilizing neural network against Vanilla and Enhanced MCTS. In Midrange Shaman mirror matches the approach underperformed with about 20% win rate in comparison with the other approaches, however despite the reduction of MCTS parameters its performance in Pirate Warrior mirror matches is not as poor as expected, winning approximately 40% of games. Against Freeze Mage its performance is similar to the Enhanced MCTS and a slight improvement can be observed against Vanilla MCTS.

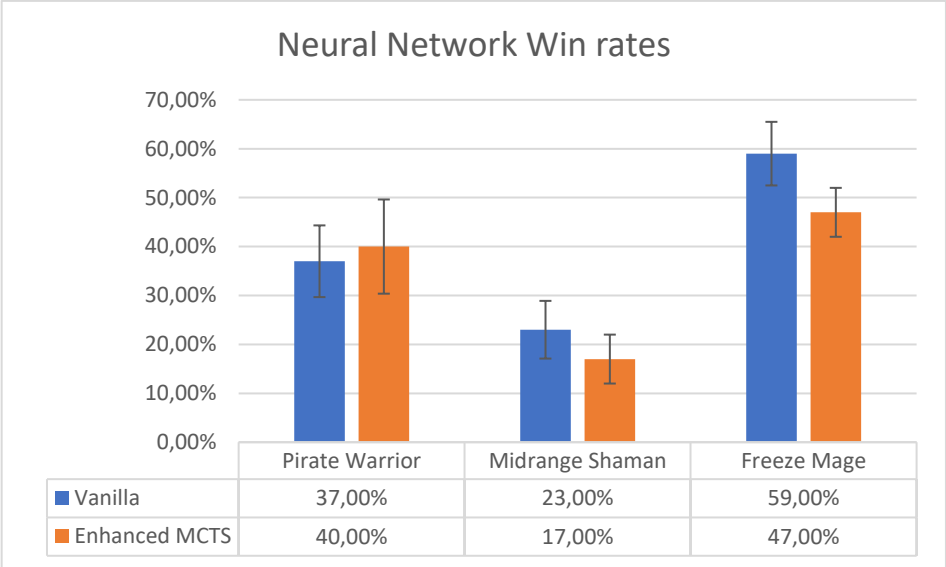


Figure 15 – Win rates of depth limit, bucketing and NN

In Figure 16 are the win rates of MCTS utilizing only depth limited search and bucketing against other implemented approaches. The artificial player shows dominant performance with Pirate Warrior (56.0% against Vanilla and 70.0% against Enhanced) and Midrange Shaman (64.4% against Vanilla and 56.0% against Enhanced), on the other hand its results in Freeze Mage mirror matches are poor as was expected based on observations from previous experiments, winning 24.8% against Vanilla and 22.0% of games against Enhanced. However, according to the results of MCTS with neural network matches, performance of this approach can be significantly improved if an action selector was incorporated into its rollout decision-making.

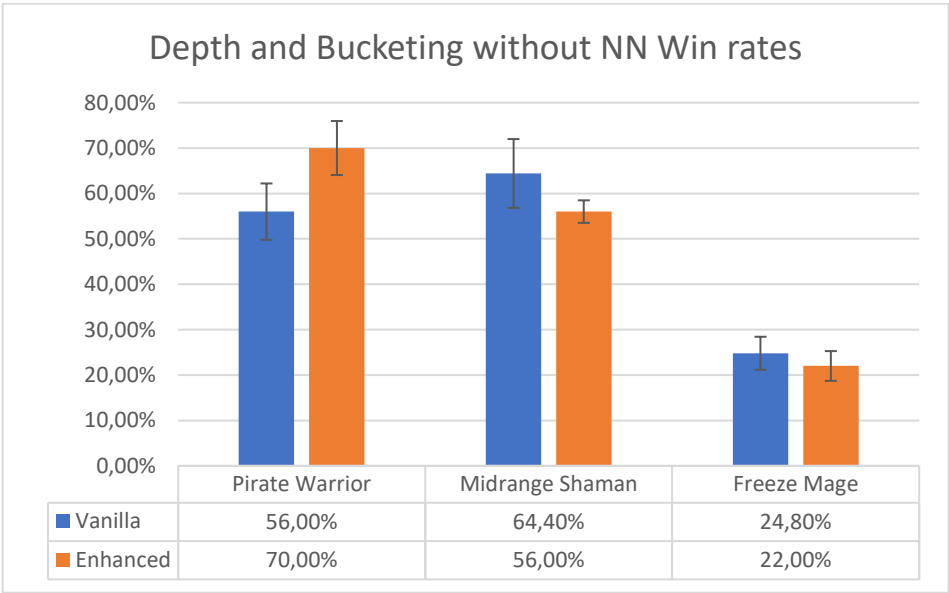


Figure 16 – Win rates of depth limit and bucketing without NN

Figure 17 shows results of matches between Vanilla MCTS and its Enhanced version. The performance of both approaches is similar in Pirate Warrior mirror match, but against Midrange shaman a slight improvement can be observed and in Freeze Mage matchup the enhanced variation wins 74.40% of games. It is safe to say that the addition of greedy heuristic-oriented action selector to MCTS rollouts and the deck database to determine unknown information improved the performance of Vanilla MCTS, especially since the enhanced version needs 2.3 less thinking time to make a decision with its current configuration than its Vanilla counterpart.

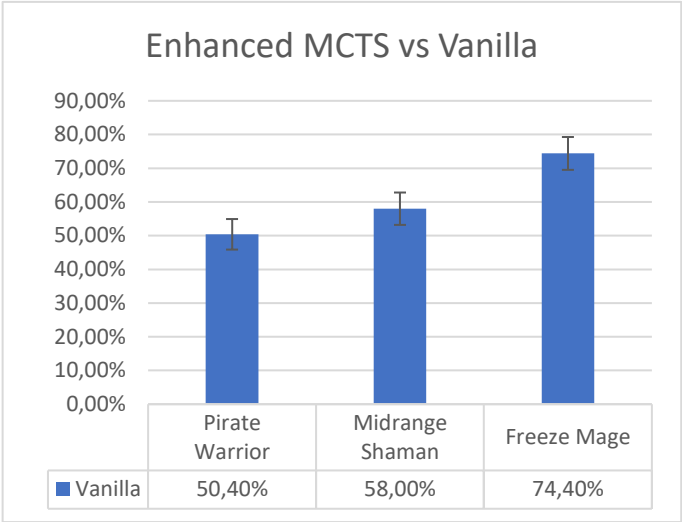


Figure 17 – Win rates of Enhanced MCTS over Vanilla MCTS

8.6 Summary

In this section, experiments with the implemented approaches were conducted. At first, the parameter selection for the Vanilla MCTS was discussed, finalizing its configuration with 500 iterations, 11 worlds and 0.7 exploration parameter. Afterwards it was matched against artificial players available in the Metastone simulator. The approach won every game against random oriented agent and showed dominant performance over Greedy player, however it underperformed against the GSV behavior. Its enhanced variation achieved even higher win rates against Greedy behavior and won 88.4% of matches against GSV in Freeze Mage mirror match. However, just like the Vanilla MCTS it was unable to reliably defeat the GSV player with Pirate Warrior and Midrange Shaman decks. Then, the second improvement of the Vanilla MCTS was matched against the Metastone players. With the addition of depth limited search and card draw chance event bucketing the approach achieved the best performance against GSV in Pirate Warrior and Midrange Shaman mirror matches. However, win rate against Freeze Mage was significantly reduced. Afterwards the neural network was integrated into the approach, but the evaluation of the network proved to be time consuming and the parameters of MCTS and range of the experiments had to be decreased. This had negative effect on the performance of the approach in Pirate Warrior

and Midrange Shaman matches, however against Freeze Mage the win rate increased significantly, showing that the addition of action selector to rollouts can have beneficial effects on the players performance despite the parameter reduction. Finally, the implemented approaches were matched against each other. Enhanced MCTS achieved the best results in Freeze Mage mirror matches, however MCTS with depth limited search and bucketing was the most successful in Pirate Warrior and Midrange Shaman games. According to statistics available in MetaCentrum a total of 351 jobs were computed in the cluster with the total of 518.8 CPU days spent.

Results of most of the matches are summarized in tables 7 (Pirate Warrior), 8 (Midrange Shaman) and 9 (Freeze Mage):

	GSV	**Vanilla**	**Enhanced**	**No-Neural**	**Neural**
Vanilla	31.2%	-	49.6%	44.0%	63%
Enhanced	36.0%	50.4%	-	30.0%	60.0%
No-Neural	56.8%	56.0%	70.0%	-	-
Neural	18.0%	37.0%	40.0%	-	-

Table 7 – Pirate Warrior mirror matches

	GSV	**Vanilla**	**Enhanced**	**No-Neural**	**Neural**
Vanilla	37.2%	-	42.0%	35.6%	77.0%
Enhanced	39.2%	58.0%	-	44.0%	83.0%
No-Neural	44.8%	64.4%	56.0%	-	-
Neural	20.0%	23.0%	17.0%	-	-

Table 8 – Midrange Shaman mirror matches

	GSV	**Vanilla**	**Enhanced**	**No-Neural**	**Neural**
Vanilla	66.4%	-	25.6%	75.2%	41.0%
Enhanced	88.4%	74.4%	-	78.0%	53.0%
No-Neural	44.0%	24.8%	22.0%	-	-
Neural	79.0%	59,0%	47.0%	-	-

Table 9 – Freeze Mage mirror matches

9 Conclusion and Future Work

This chapter concludes the thesis and gives few suggestions for future work.

9.1 Conclusion

In this thesis the collectible card games and challenges they represent for the development of artificial intelligence were introduced. Afterwards CCG Hearthstone was presented as testbed for the conducted research and experiments, utilizing an open-source simulator called Metastone that offers variety of artificial players.

To play Hearthstone, Vanilla MCTS was designed utilizing Perfect Information approach to deal with the unknown information. This approach required 27 seconds to decide and reliably defeats random playing player, winning every match, as well as prevailing over the greedy heuristic oriented agent with 80% average win rate. The approach also achieved 66.4% win rate in Freeze Mage mirror matches against GameStateValue behavior, which is the strongest artificial player offered by Metastone, however it underperformed against GSV in Pirate Warrior and Midrange Shaman matches, winning 31.2% and 37.2% of games played respectively. Two improved variants of the Vanilla MCTS are designed according to existing works.

The first is the Enhanced MCTS variation, utilizing deck database to determine hidden information and rollout driven by greedy action selector with respect to a heuristic. The improvements increased the win rates of Vanilla MCTS in all matches, winning 88,4% of games against GSV in Freeze Mage mirror match. However, in matches with other decks the Enhanced MCTS underperformed with win rates slightly below 40%. Based on results of matches with Metastone players as well as games played with Vanilla MCTS the enhancements truly improved the performance of the previous method while requiring only 11.67 seconds of thinking time with its configuration.

The second improvement utilized depth limited search, chance event bucketing applied to card draw events that samples some of the possible outcomes during construction of the search tree, and machine learned card play policy to guide rollouts. The addition of depth limit and bucketing accomplished the best results with only 4.41 seconds thinking time against GSV in Pirate Warrior and Midrange Shaman mirror matches with win rates 56.8%

and 44.8% respectively but its performance against Freeze Mage was significantly reduced from 66.4% to 44.0%.

The neural network showed disappointing performance with only 32.61% accuracy and an average evaluation time of 43.85 milliseconds. With the original MCTS configuration about 44 000 evaluations are needed to decide, requiring nearly 32 minutes to select an action. Because of the slow evaluation the MCTS parameters had to be toned down to perform the experiments. The final approach required 3 minutes to decide but performed poorly with Pirate Warrior and Midrange Shaman decks against the GSV, winning less than 20% of matches, as well as Vanilla and Enhanced MCTS with about 40% win rate against Pirate Warrior and 20% win rate with Midrange Shaman. However, its results with Freeze Mage deck show that the performance of depth limited MCTS with bucketing can be significantly improved with the addition of fast and reasonable action selector to guide rollouts, achieving 79% win rate against GSV, 59% against Vanilla MCTS and 47% in Enhanced MCTS matches.

To perform the experiments a computational cluster MetaCentrum was utilized. According to the statistics offered by MetaCentrum a total of 351 jobs were computed with 518.8 CPU days spent with the evaluations.

9.2 Future Work

The application of the neural network could be revisited in order to lower its time requirements for evaluations and increase its accuracy by choosing a different network architecture or designing a new way to encode the game states. With a fast and effective action selector during rollouts the depth limited MCTS with bucketing could achieve high playing strength.

The bucketing MCTS employs randomized determinization of the hidden information. In order to create better determinization of the hidden information the deck database used by the Enhanced MCTS could be incorporated, or a completely different method could be utilized as well. For example, a machine learned method that takes an encoding describing cards played by the opponent player as an input and for each card outputs a probability with which the card could appear in opponent's deck. The deck would then be filled with the most probable cards. Other possibility would be to keep track of currently popular cards among players, considering only the cards frequently used. This could significantly decrease

number of cards available for determinization because in every CCG there is a high amount of cards that are generally weak in comparison with others and are almost never seen played. Replacement of PIMC with ISMCTS could also be an interesting subject for further examination.

The determinization method of the Enhanced MCTS could be improved as well. If the database is regularly updated the bot will be able to accurately determine the opponent's deck. However, opponent's hand and secrets are guessed randomly. Each deck could be assigned with a strategy it employs and based on the strategy each card could have a different probability of being in opponent's hand. For example, if the opponent plays an aggressive deck with a few high manacost cards, the higher probabilities could be assigned to lower manacost cards to mimic the aggressive gameplay and increase the probability of higher cost cards with each passed turn.

During the game of Hearthstone the effect of some actions might depend on previously executed actions, however that is not always the case. For some action sequences it does not matter in which order they are performed and always lead to the same state. For a sequence of actions that leads to the same state no matter the order they are executed in it is inefficient to explore each of the possible action permutations. Instead, a single possible permutation can be selected to represent the action sequence and the remaining permutations are not considered. This allows MCTS to get better estimation of the score for the selected sequence, using the time budget more efficiently.

Each of the proposed approaches had unlimited thinking time, but in Hearthstone player's turn is limited to 75 seconds. During player's turn it is unknown how many actions the player will perform, making it hard to distribute the available time budget. Approaches to reduce the time complexity of implemented agents and a method that distributes the available time budget to agent's decision-making process could be explored in future work.

10 Bibliography

[1] jleclanche, „fireplace“, https://github.com/jleclanche/fireplace

[2] HearthSim community, https://hearthsim.info/

[3] danielyule, „hearthbreaker“, https://github.com/danielyule/hearthbreaker

[4] oyachai, „HearthSim“, https://github.com/oyachai/HearthSim

[5] utilForever, „HearthStonepp“, https://github.com/utilForever/Hearthstonepp

[6] demilich1, „Metastone“, https://github.com/demilich1/metastone

[7] Guillaume Chaslot, Sander Bakkes, Istvan SzitaandPieter Spronck, „Monte-Carlo Tree Search: A New Framework for Game AI“, Proceedings of the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference, pp 216-217

[8] G. Chaslot, J.-T. Saito, B. Bouzy, J. Uiterwijk, and H. J. Van Den Herik, “Monte Carlo strategies for computer Go,” in Proceedings of the 18th BeNeLux Conference on Artificial Intelligence, Namur, Belgium, 2006, pp. 83–91.

[9] P. Auer, N. Cesa-Bianchi, and P. Fisher, “Finite-time analysis of the multiarmed bandit problem,” Machine Learning, vol. 47, pp. 235–256, 2002.

[10] L. Kocsis and C. Szepesvari, “Bandit based Monte-Carlo planning,” in Proc. 17th Eur. Conf. Machine Learning, 2006, pp. 282–293.

[11] Eckart Zitzler, Evolutionary, „Algorithms for Multiobjective Optimization: Methods and Applications“

[12] P. García Sánchez, A. Tonda, G. Squillero, A. M. Mora and J. J. Merelo, “Evolutionary Deckbuilding in HearthStone”, 2016 IEEE Conference on Computational Intelligence and Games (CIG), Santorini, Greece, 2016. To appear. doi: To Appear.

[13] André Santos, Pedro A. Santos, Francisco S. Melo, “Monte Carlo Tree Search Experiments in Hearthstone”, IEEE Conference on Computational Intelligence and Games 2017, pp. 272-279

[14] https://www.andremlsantos.com/research

[15] E. Bursztein, “I am a legend: Hacking “Hearthstone” using statistical learning methods,” in Proc. 2016 IEEE Int. Conf. Computational Intelligence in Games, 2016.

[16] Andrzej Janusz, Tomasz Tajmajer, Maciej Świechowski “Helping AI to Play Hearthstone: AAIA’17 Data Mining Challenge”

[17] Łukasz Grad, “Helping AI to Play Hearthstone using Neural Networks”, Proceedings of the Federated Conference on Computer Science and Information Systems pp. 131–134

[18] Evgeny Patekha, “Application of machine learning to help AI to play Hearthstone”, Communication papers of the Federated Conference on Computer Science and Information Systems, pp. 45–48

[19] André Santos, Pedro A. Santos, Francisco S. Melo, “Monte Carlo Tree Search Experiments in Hearthstone“, IEEE Conference on Computational Intelligence and Games 2017, pp. 272-279

[20] G. Chaslot, M. Winands, H. van den Herik, J. Uiterwijk, and B. Bouzy, “Progressive strategies for Monte-Carlo tree search,” New Mathematics and Natural Computation, vol. 4, no. 3, pp. 343–357, 2008.

[21] B. Miller and D. Goldberg, “Genetic algorithms, tournament selection, and the effects of noise,” Complex Systems, vol. 9, pp. 193–212., 1995.

[22] Maciej Świechowski, Tomasz Tajmajer, Andrzej Janusz, „Improving Hearthstone AI by Combining MCTS and Supervised Learning Algorithms“

[23] J. R. Long, N. R. Sturtevant, M. Buro, and T. Furtak, “Understanding the success of perfect information monte carlo sampling in game tree search.” in AAAI, 2010.

[24] P. I. Cowling, E. J. Powley, and D. Whitehouse, “Information set monte carlo tree search,” IEEE Transactions on Computational Intelligence and AI in Games, vol. 4, no. 2, pp. 120–143, 2012.

[25] Shuyi Zhang, Michael Buro, “Improving Hearthstone AI by Learning High-Level Rollout Policies and Bucketing Chance Node Events”

[26] Ilya Kachalsky, Ilya Zakirzyanov, Vladimir Ulyantsev ,“Applying reinforcement learning and supervised learning techniques to play Hearthstone”

[27] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, “Playing atari with deep reinforcement learning,” arXiv preprint arXiv:1312.5602, 2013.

[28] J. Zhu, “Will our new robot overlords play Hearthstone with us?,” CS 229 Final Report, 2016.

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

[30] https://metavo.metacentrum.cz/en/index.html

[31] https://www.metacentrum.cz/en/

[32] https://www.egi.eu/

[33] Michael Nielsen, “Neural Networks and Deep Learning”, 2018.

11 Appendix

11.1 Predicting Opponent's Deck

Before the match begins each player constructs a deck of 30 cards from over 1000 available cards and the selected deck is unknown to the opponent. Being able to predict the content of the opponent's deck can be beneficial for the player as it allows them to adjust their strategy accordingly.

Ellie Bursztein proposed in his work [15] to utilize a neural network to predict the opponent's deck. According to his work there are multiple reason why deck construction is predictable:

1. Some cards are restricted to certain heroes. Therefore, Holy Light, which is a spell specific to Paladin, can't be a part of a Warlock's deck
2. Some cards are designed to work well with each other (synergize)
3. Some cards are simply weaker in comparison with a different card with the same manacost and are never seen played in competitive scene
4. Netdecking. A process where players attempt to replicate successful decks (a winning deck in a tournament or a deck used by a professional streamer). These decks become popular among the player base and are often encountered while playing the game.

The method is based on machine learned ranking system that models relations between cards as a set of bigrams (a sequence of two adjacent elements). The learning data consists of a set of game replays with sequence of cards played by the opponent during the match. For each replay a combination of all possible bigrams of played cards is constructed. These bigrams are then used to construct an occurrence table where each bigram is assigned with the number of games in which these two cards were played together (regardless of order).

During the game, whenever opponent plays a card, the occurrence table is searched for all bigrams that contain the played card. This way a set of cards that co-occurred with the played card is received. This is done for each card played by the opponent during the game. All received sets are then added together into one set (therefore, if two different sets contained the same card, the co-occurrence numbers of the card are summed together) and the cards with the highest number of co-occurrences are the most probable to be contained in the opponent's deck (Figure).

Figure - Card prediction example

Image adopted from [15]. Source Hearthstone

11.2 Automated Deckbuilding

Deck creation is an important aspect of collectible card games. It is typical for decks to have its strengths and its weaknesses. However, some decks are in general performing better than other decks, which leads to establishment of a so called “meta”. Meta is a set of the most popular decks in the current environment and these decks are often encountered while playing the game. But each deck has its weaknesses that can be exploited by other decks to increase their performance against them. The win rate of the targeted deck then begins to decline and other decks replace it at the top of the ladder. This represents an interesting challenge as it forms a changing environment in which the decks have to be constantly adapting to the current situation

The matter is examined in [12] where an evolutionary algorithm is suggested to create an optimal deck against a specific meta. The evolutionary algorithm follows steps described in section 4.2. An initial set of random decks is created and their quality is described by a fitness function. This function considers three parameters:

1. **Correctness:** Each deck has to consists of exactly 30 cards and can have up to 2 copies of a non-legendary card and 1 copy of a legendary rarity card. If the deck breaks these rules its fitness value is set to minimum.

2. **Victories:** Each deck plays 16 simulation in Metastone simulator against each targeted deck in the current meta. The value is equal to the total number of victories achieved and should be as high as possible
3. **Standard Deviation:** The goal of the deck is to perform well against all targeted decks and not only few of them. After evaluating, the standard deviation of achieved victories is computed. This value is to be minimized.

This method was used to create optimal decks against a set of human-made decks that were considered to be strong at the time of the research. Two experiments were conducted with two resulting decks: A Hunter deck and a Mage deck. These decks were able to perform extraordinarily well against some targeted decks (evolved Hunter deck was able to win all 16 games against targeted druid deck) but were underperforming against others (37.5% win rate of evolved hunter against targeted priest deck).