FABIAN MOIK

TITLE

Bachelor's Thesis

to achieve the university degree of

Bachelor of Science

Bachelors's degree programme: Information and Computer Engineering

submitted to

Graz University of Technology

Supervisor

Assoc.Prof. Dipl.-Ing. Dr.mont. Pernkopf Franz

Graz, Oktober 2018

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present bachelor's thesis.

Date	Signature

Abstract

This is a placeholder for the abstract.

Contents

List of Figures

List of Tables

1 Introduction

Until recently artificial intelligence (AI) research has focused on creating intelligent algorithms capable of winning against the best human players in a variety of board games such as *chess*, *backgammon* or *go*. In these board games players have perfect information about the entire game state at any given moment, which allows algorithms to brute-force calculate an optimal strategy. Card games, such as *poker* however do not provide perfect information about the game state and therefore provide a much more challenging environment for intelligent algorithms. The game of poker has several interesting characteristics including *imperfect information*, *risk management*, *opponent modeling* and *deception*, that make it an ideal candidate for AI research. Developing a world class artificial poker agent may also help solving open questions in AI research and provide important research benefits. [**?**, **?**].

The goal of this thesis is it to develop an artificial poker agent capable of understanding the fundament concepts of *No-Limit Texas Hold'em poker*, to establish a profitable strategy against *static opponents*, by applying *evolutionary*

algorithms to an artificial neural network.

Chapter 2 of this thesis gives an introduction to the game of poker, focusing on the *No-Limit* variant of *Texas Hold'em*, and explains the basic rules of the game. Chapter 3 provides an overview of previous work done in the field of computer poker, especially focusing on the creation of artificial poker bots. Chapter 4 discusses the design and implementation of the evolutionary neural network poker bot and explains some key components to achieve a high level poker skill. Moreover the procedure of training and evaluating the neural network agents is explained in detail in this chapter. In Chapter 5 experimental results and interesting observations are discussed. Chapter 6 summarizes conclusions taken from this work and provides possible future work in this field.

2 Poker Basics

2.1 What is Poker

In David Sklansky's book [**?**] poker is being described as a term for a whole family of card games, which can be grouped into different types. In some game variants such as *Texas Hold'em or Omaha* the player with the highest ***hand rank*** [1] wins, in others such as *Razz or Badugi* the player with the lowest hand rank wins, and in yet others the ***pot*** is split between the player with the highest and the player with the lowest hand rank [**?**].

Among all poker variants there are different ***betting*** structures. Many poker formats offer ***limit*** variants of the game and ***no-limit*** variants. In limit games the ***bet size*** has an upper and lower limit, whereas in no-limit games the bet size is just restricted by the amount a player has currently left in his ***stack*** [**?**]. While bet sizes may not be restricted in no-limit games there are

[1] All poker specific terms mentioned the first time, appear in bold, italics throughout this bachelor's thesis. They are defined and briefly explained in the Appendix A.

specific rules in all poker variants that determine the minimum bet size at any given time in the game regardless of the variant of the game itself.

2.2 No-Limit (NL) Texas Hold'em

The focus of this work is drawn to the no-limit version of the Texas Hold'em poker variant. Not only is it the most widely played poker variant but also considered to be the most challenging form of poker [**?**]. While finding the optimal strategy for Texas Hold'em poker may be complex it has exceptionally simple rules [**?**].

2.2.1 Poker Rules

A Texas Hold'em ***hand*** begins in a stage called ***pre-flop***. In this stage every player at the table is dealt two cards. The cards are dealt face down and are exclusive to the player receiving the cards [**?**]. Each seat at the table has a special meaning throughout a poker hand. The seat (also called ***position***) which owns the ***dealer-button*** determines the two positions that have to place ***forced bets*** called the ***small blind*** and the ***big blind***. This dealer-button moves clockwise by one position once a whole poker hand is finished. The player directly to the left of the ***dealer***, the player currently holding the dealer-button, is forced to bet one small blind and the player two seats to the left of the dealer has to bet one big blind, the equivalent of two small

blinds [**?**]. These bets serve the purpose of preventing players to wait for the best hand without any punishment.

In case of a Texas Hold'em poker tournament most often another type of forced bet called the ***ante*** is present in every hand of the tournament [**?**]. It usually is a percentage portion of the current big blind in the range of 10%-20% that has to be contributed to the pot by each player. In poker tournaments the small blind and the big blind are incremented after a fixed time interval to advance the game.

The pre-flop stage is concluded once each player has acted in the first betting round. The next stage, called the ***flop***, is introduced by three face up dealt cards to the ***community board*** and players once again have the chance to bet in this round. The third stage is called the ***turn*** and a fourth card is dealt to the community board. After concluding the third round of betting a fifth ***community card*** is dealt on the ***river*** and players are allowed to bet one last time. If there is more then one player left after this final round of betting the cards of all remaining players are revealed in a so-called ***showdown***. The best combination of two player's ***hole cards*** and three community cards decides the winner. If two players have the same hand rank, the pot is split between both of them [**?**].

2.2.2 Betting

Sophisticated betting strategies are the number one key component for succeeding in the game of NL Texas Hold'em poker. Choosing the optimal bet size in any given situation allows players to maximize their winnings and at the same time minimize their losses [**?**].

Each betting round is either opened by a bet or a ***check***. In a clockwise manner players may then decide to check or bet if there was no bet placed yet. If a bet has already been placed, the remaining players may either call the bet, ***raise*** the bet or ***fold*** to that bet. In NL Texas Hold'em the bet or raise amount is only limited by the remaining ***chips*** a player has [**?**].

Betting Options [?, ?]

- **Check / Call**: A check describes the action of staying in the hand without committing any money to the pot. To check a hand there must be no previous bets in the current betting round. By calling someone's bet a player wagers the amount of chips to equalize the amount of chips committed to the pot by each player.

- **Bet / Raise**: A bet describes the action of a player being first to put a certain amount of money into the pot. If there already exist a bet a player may still put more money into the pot then the previous bets

did. This is referred to as a raise.

- **Fold**: Folding a hand means no longer being interest in the hand and not willing to put any more money into the pot. If however no bet has been placed in the current betting round yet, a player may rather check his hand instead of folding it.

2.2.3 Hand Rankings

Texas Hold'em poker is a poker variant where the highest ranking 5-card combination wins [**?**]. To decide the winner of a poker hand the best possible 5-card combination of each remaining player is compared against each other [**?**]. The ranking system of 5-card combinations can be found in Table **??**.

Hand rank and sample hand	Description
Royal Flush A♠ K♠ Q♠ J♠ 10♠	Highest ranking Flush plus Straight
Straight Flush J♣ 10♣ 9♣ 8♣ 7♣	5 cards of same suit and in sequence, without the Ace
Four of a Kind Q♣ Q♥ Q♦ Q♠ 2♠	Four matching cards of same rank
Full House Q♣ Q♥ Q♦ 2♣ 2♠	Three of a kind and one pair
Flush 7♣ Q♣ 9♣ 2♣ K♣	5 cards of same suit, not in sequence
Straight 7♣ 8♦ 9♥ 10♣ J♣	5 cards in sequence without matching suits
Three of a Kind Q♣ Q♦ Q♥ 8♥ 2♣	Three cards of the same rank, two unmatched cards
Two Pair 7♣ 7♦ 9♦ 8♥ 8♣	Two pairs of two matched cards
One Pair 7♣ 10♦ 9♦ 8♥ 8♣	Two cards of the same rank, three unmatched cards
High Card 7♣ 10♦ 9♦ A♥ Q♣	No matches between 5 cards. The highest card counts.

Table 2.1: Hand rankings for 5-card combinations (strongest to weakest)

3 State-Of-The-Art

The game of poker has many interesting properties which proved to offer a challenging test environment for artificial intelligence research. Over the last decade scientists and researchers have studied the game from a game theoretical view to find optimal solutions but also tried a number of machine learning algorithms and artificial intelligence systems on the game [**?**, **?**]. A lot of research though focused on simplified versions of the game because these variants of poker are easier to analyze but still offer demonstrations of game theoretical principles [**?**]. Nonetheless over the past few years, abstract versions of poker have been used to first successfully train algorithms on this simple version of the game and then transfer the obtained knowledge to a full version of the game [**?**, **?**].

3.1 Knowledge-based Poker Agents

Knowledge-based systems can be broken into two categories, namely **rule-based expert systems** and **formula-based methods**. In general knowledge-

based systems require the knowledge of an expert player to design the system [**?**]. A *rule-based expert system* in its simplest form is a sequence of if-else statements for frequently occurring scenarios of the game which determine the desirable ***betting action***. More advanced human players describe their poker hands in a very similar way when they break them down in a discussion [**?**]. *Formula-based methods* on the other hand try to generalize the problem by defining a formula that takes a set of inputs and outputs a so-called ***probability triple*** upon which a betting decision is made. Inputs to the formula may describe important information about the current game state and can be weighted to reinforce the importance of certain inputs over others [**?**].

While rule-based expert systems may yield reasonably good results in the first stage of a poker hand (pre-flop), they fail to shine in later stages of the game because they become increasingly difficult to maintain with additional and sometimes even conflicting information added at each stage. Furthermore static strategies are prone to exploitation and therefore not competitive with other approaches [**?**].

3.2 Simulation-based Poker Agents

In general simulation based methods rely on repeatedly simulating an outcome in order to approximate the resulting expected value of an action [**?**]. A frequently used simulation method applied to the game of poker is

the so called *Monte-Carlo simulation* or *Monte-Carlo Tree Search*, which is a procedure that searches the game tree by sampling the possible choices in a game state and simulates the action taken to the bottom of the tree. By repeating this procedure a robust expectation value can be computed [**?**]. In Billings *et. al* [**?**], the technique of *selective sampling* is described, which uses information about the opponents to bias the selection of possible card combinations a player may hold. With this modification to the sampling algorithm they were able to create a more dynamic betting strategy due to the gain of valuable information about opponents.

3.3 Game-Theoretic Optimal Poker Agents

The currently best performing poker-playing programs are approximating a Nash equilibrium [**?**]. In game theory a Nash equilibrium describes a state of a game where no player can find an action that would yield a better outcome than the suggested equilibrium action, given that all other players also choose to take the suggested action [**?**].

This strategy has proven to achieve great results in zero-sum games, like NL ***heads-up*** poker [**?**]. Most successful heads-up poker bots use an abstract version of the poker variant in which they approximate the Nash equilibrium and then transfer the decision made in the abstract version to the real version of the game. One of the most successful algorithm in approximating the Nash equilibrium in an abstract version of the game is called *Counterfactual*

Regret Minimization. The top three poker bots in the *Annual Computer Poker Competition* (ACPC) 2016 used a variant of the Counterfactual Regret (CFR) algorithm to defeat their competitors [**?**].

3.4 Adaptive / Exploitive Poker Agents

Nash equilibrium approaches and other static poker strategies are vulnerable to exploitation [**?**]. Adaptive strategies try to tackle this problem by quickly adapting to the playing-style of the opponents and exploiting their weaknesses. Two algorithms, namely the *Miximax* and *Miximix* algorithm achieve this by searching an adaptive imperfect information game tree. Decisions are then made by considering a *randomized mixed strategy* associated with the decision node of the searched tree [**?**].

3.5 Bayesian Poker Agents and Evolutionary Algorithms

3.5.1 Bayesian Poker Agents

A Bayesian network is a probabilistic graphical model. Each node in the directed acyclic graph represents a random variable and edges between nodes represent the conditional dependencies of variables. Nodes are associated

with a probability function which returns the conditional probability values based on their parent's values. A probability distribution over the random variables can be retrieved by propagating the probabilities of initialized nodes throughout the network [**?**].

Compared to other poker playing agents, bayesian agents performed badly in the AAAI Computer Poker Competitions in the last years. There is still a lot of room for improvement in Bayesian based networks and further research in this field has to be done to be competitive in upcoming competitions [**?**].

3.5.2 Evolutionary Algorithms

Evolutionary algorithms try to mimic the evolution process of biological bodies. Agents of one generation are competing in a population to achieve a good score for a given task. The best agents are then evaluated and chosen as parents for the next generation. Some small changes are applied to the genetic information of the newly created offspring before they compete with the best agents of the previous generation in a new population. This process repeats for a number of generations and optimally should yield a global increase of performance [**?**].

Over the last years evolutionary algorithms were used in some studies to train artificial neural networks to play poker. Nicolai and Hilderman [**?**] train neural network agents to learn the game of poker and propose some

methods to counter inherent problems of evolutionary algorithms. While poker agents trained with an evolutionary algorithm have not yet performed very well against other poker bots, there is still a lot to learn and discover from evolutionary algorithms applied to neural networks in the future [**?**].

4 Implementation - Structure of the Bot

In this chapter the architecture of both the test environment and the poker playing agents is described and the learning process of the poker agents is explained. There are some key components that distinguish a strong poker player from a weak one. A well defined *betting strategy* both attuned to the mathematical principles such as *hand strength (HS)* and *hand potential (HP)* and to the interpretation of opponent's tendencies decides on the profitability of a poker player over the ***long run*** [**?**]. This chapter describes a possible way to implement and apply these concepts to create a poker playing agent, capable of understanding the situation on the table and assessing its hand strength versus opponents.

4.1 Architecture of Testbed and Neural Network Agents

Although there are a handful open source poker testbeds on the internet, none of them has proven to be suitable to achieve the goal of this thesis. Some of them lack the needed flexibility to extract all necessary information for the bot out of the current game state, others are not designed to run tournament poker simulations but only ***cash games*** or other poker variants. Therefore in this thesis a testbed was crafted, which can be arbitrarily modified to the needs of the user. However, the main focus of this thesis was on the creation of a test environment that can simulate millions of poker tournaments in as little time as possible. At the same time it should provide our poker agents with all the needed information about the current game state and opponents at the table. The poker testbed was written in *C++* due to little performance overhead at runtime and its speed and efficiency.

4.1.1 Poker Testbed

The basic structure of the poker test environment, referred to as *testbed* in this context consists of following components:

- ▷ Game object
- ▷ Table object
- ▷ Dealer object
- ▷ Player object
- ▷ Artificial Intelligence (AI) object
- ▷ Deck object
- ▷ Card object
- ▷ Rules object

To run an arbitrary number of simulated tournaments with one generation of agents, a *game* object is initialized. The *rules* for the game, including the ***Buy-In***, the total number of players, the maximum number of players per table and the ***blind structure***, are then passed to the game object and all participating *players* are added to the game.

Depending on the number of players participating in the tournament a number of *table* objects are created. The game object then distributes all players across the tables and places one *dealer* on each table. Dealers are not participating in the tournament but their job is it to deal cards to the players, apply forced bets (*Small Blind*, *Big Blind* and *Antes*), deal community cards, determine the winning player for each hand and split the pot accordingly. Each dealer holds a *deck* object which consists of 52 unique cards, 13 cards for each suit. The *cards* themselves are also objects and contain the information about their ***suit*** and the card's value. They also offer some convenience functions for representing the card's value in a readable format for the human.

Each player holds an *AI* object and is assigned a unique ID. The dealer tells the player when to act and provides him with all the public information

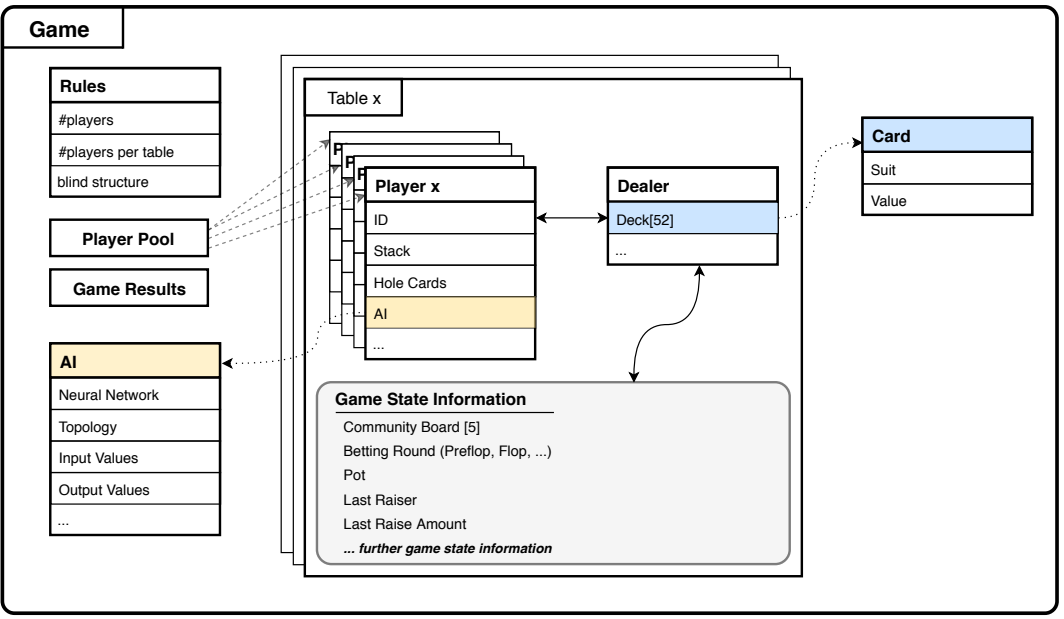

Figure 4.1: High Level Architecture of Testbed.

about the game state. The player's AI then decides on the basis of this information how it would act in this situation and the player executes this action. In case of a betting action the player selects an appropriate bet size according to a betting strategy and places it on the table. An in depth look into the architecture of the bot is provided in subsection **??**.

Figure **??** shows the high level architecture of the described testbed. Boxes represent objects, list items within the box represent properties of the corresponding object. Bidirectional arrows indicate that information is exchanged in both directions between two objects. Dotted arrows zoom in on a property of an object. Rounded boxes represent a collection of properties of an object.

4.1.2 Neural Network Agents

In the conducted test series of simulating hundreds of tournaments in a single population of poker agents, evolving neural networks were used to train the agents on the game of NL Hold'em poker. An agent in this population is represented by the *player object*, described in **??**. The player object holds an *AI object* which implements a neural network and returns the desired action to be taken, given the public information about the current game state. This public game state information is provided by the *dealer object*.

Structure of the Neural Network

The agents to be examined are implemented as fully connected feed-forward neural networks with a network topology of 24-14-3, which corresponds to 24 *input neurons* in the input layer, 14 *hidden neurons* in the hidden layer and three *output neurons* in the output layer [**?**]. For the hidden layer a **sigmoid** activation function is used and a **softmax** activation is performed at the output layer. The three output neurons represent the so called **probability triple** (f, c, r), which specifies the probability distribution for the actions *fold*, *check/call* or *bet/raise* at the current state of the game [**?**]. The full architecture of the neural network with all its input features can be seen in Figure **??**. The selection of input features was strongly influence by the work of Nicolai in [**?**] and are described in the sequel.

The final structure has not yet been determined (depends on opponent modeling module)

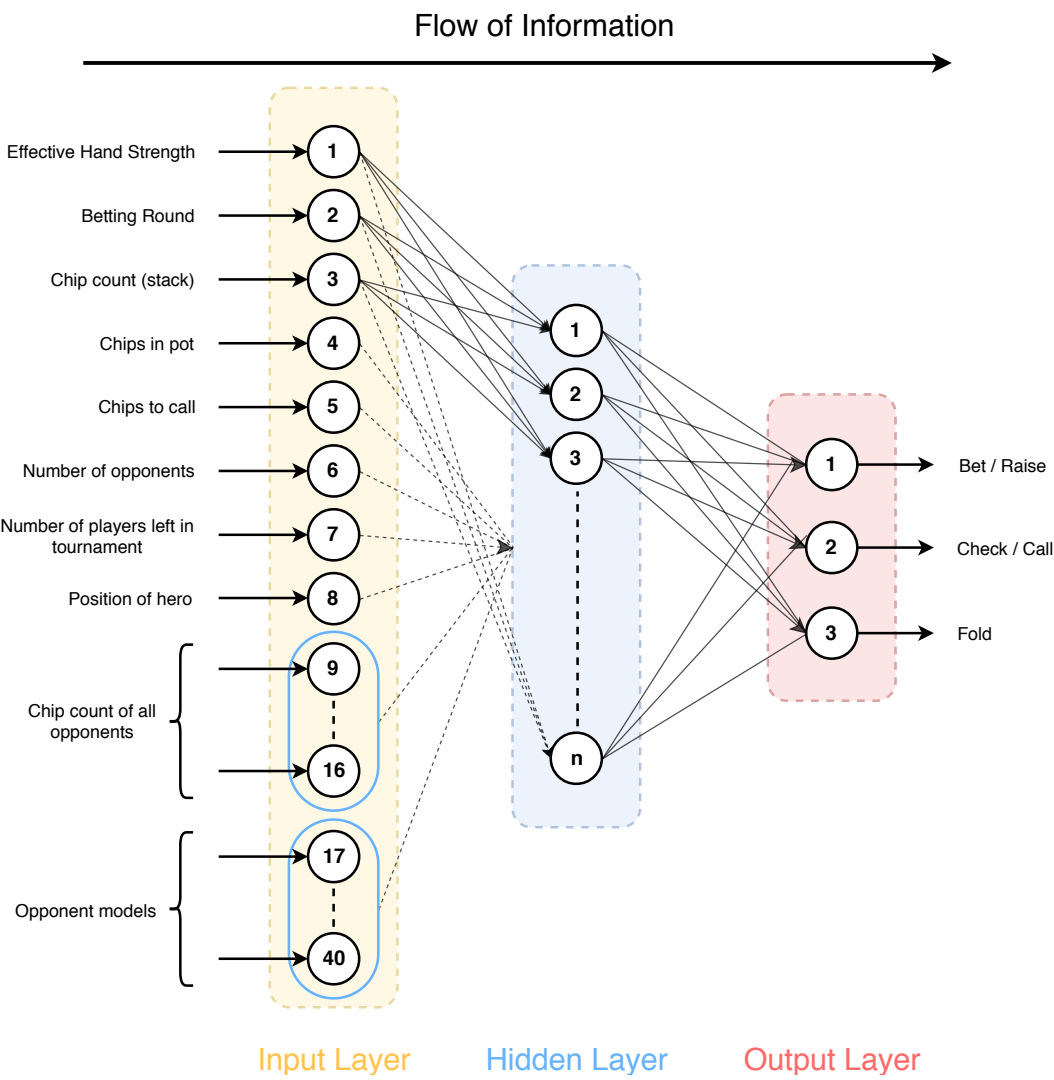

Figure 4.2: Architecture of the neural network with all the input features

Effective Hand Strength (EHS)

The EHS is an indicator for how likely a hand is winning at showdown considering the current hand strength but also the positive and negative hand potential [**?**]. In conjunction with other components the EHS should be used to help selecting a suitable betting action. A more detailed description of EHS can be found in Subsection **??**.

Betting round

The second input for the neural network is the current betting round in the game. This can be PREFLOP, FLOP, TURN or RIVER. The betting round is an important property because a good preflop strategy varies from a good postflop strategy and therefore this property should not be withheld from the neural network.

Chip count

The next feature is the number of chips the acting agent has. This represents the stack of the player.

Chips in pot

The fourth feature is the number of chips already in the pot, including all chips of previous betting rounds plus the number of chips betted by other agents in the current betting round. An agent may not be able to win all the chips in the pot if his chip count (stack) is smaller than a bet of his opponents in the current round. This means when an opponent makes a bet larger than the remaining chips in the agent's stack, the agent may only win the amount of chips that is covered by his stack. Hence the value of this feature is the effective pot size an agent can actually win when his hand is the strongest at showdown [**?**].

Chips to call

The fifth feature is the number of chips an agent has to match in order to be allowed to continue in the hand. The ratio between the amount of chips a player has to bet to stay in the hand and the amount of chips in the pot is called ***pot odds***. This is a commonly used tool in the poker community to calculate the needed winning percentage of an agent's hand to make the call a profitable play, irrespective of other factors [**?**].

Number of opponents

This feature represents the number of opponents still involved in the hand.

Number of players left in tournament

The seventh feature represents the total number of players, which are still left in the tournament. This feature is important because a strong poker player changes his strategy slightly when there are less players in the tournament. The reason being, that it is more profitable to play a tighter style once approaching the money ranks of a tournament.

Position of hero

This feature is the relative position of the agent to the dealer. The dealer position is the most valuable position in poker because the action of only two more players will follow in an unopened pot. In general it is desirable to be in a ***late position*** because many players have already acted before you and hence more information is available for the agent in late position [**?**].

Chip count of all opponents

In a ***9-handed*** multi table tournament there are at most 9 players on one table. Therefore this feature has 8 input nodes, one for each opponent on the table. The input is the number of chips an opponent has. Late in a tournament it will occur that hands are not always played 9-handed but with less then 9 players per hand. For this case the input for empty seats on the table is set to unknown and will be recognized by the neural network as

an empty seat. The first node of this 8 nodes is always the opponent directly to the left of the agent, the second node is the opponent two seats to the left of the agent [**?**].

Opponent model for all opponents

The final 24 features are used to model opponent's playing tendencies. Opponent modeling is important in the game of poker, because once a player has observed some information about the playing style of his opponents, he can adapt his own playing style to this new information. While playing against opponents in a series of tournaments there is a lot of information that can be gained about opponents. In this thesis the process of modeling tendencies of opponents focused on three fundamental statistical values. The first value describes the frequency of voluntarily putting money in the pot **(VPIP)** (also known as VP$IP). It can be calculated according to Equation **??**.

$$VPIP_{\%} = \frac{\#CALLS_P + \#RAISES_P}{\#HANDS_P} \cdot 100\% \tag{4.1}$$

$\#CALLS_P$ represents the number of hands a player has called with pre-flop, $\#RAISES_P$ represents the number of hands a player has raised with pre-flop and $\#HANDS_P$ is the total number of hands a player has played pre-flop.

Similar to the *VPIP* value the second statistical value, namely **Preflop Raise (PFR)**, shows the frequency of a player raising with a hand pre-flop instead of calling, checking or folding with it and can be calculated with Equation

??.

$$PFR_{\%} = \frac{\#RAISES_P}{\#HANDS_P} \cdot 100\% \tag{4.2}$$

The third statistical value used for opponent modeling is called **Aggression Frequency (AFq)** and can be used as an indicator for the overall aggression of an opponent.

$$AFq_{\%} = \frac{\#RAISES}{\#RAISES + \#CALLS + \#FOLDS} \cdot 100\% \tag{4.3}$$

$\#RAISES$ represent the total number of hands a player has raised with, $\#CALLS$ and $\#FOLDS$ represent the number of hands a player has called with or folded, respectively.

If those three values are considered individually, they offer little information for a poker player, but if they are seen as a whole they give a good indication about the playing style of an opponent. As a rule of thumb, the higher the gap between $VPIP$ and PFR is the more passive a player is [**?**].

4.1.3 Key Components to Achieve High Level Poker Skill

Billings et al. [**?**] described five main requirements a poker playing agent needs to fulfill in order to be competitive with the best players in the world. All these components depend on each other and need to be adjusted when a certain situation requires it. The five requirements mentioned by Billings are:

- ▷ Hand Strength

- ▷ Hand Potential
- ▷ Bluffing
- ▷ Unpredictability
- ▷ Opponent Modeling

Hand strength and **hand potential** are explained in detail in subsection **??**. In short HS and HP combined assesses the relative hand strength against opponents, ideally considering opponent-specific factors that would influence the probability of winning a hand [**?**].

To achieve this requirement a slightly different approach than recommended in [**?**, p. 208] was implemented in this study. Instead of considering opponent-specific tendencies while calculating the hand strength and hand potential, the effective hand strength was calculated under the assumption of playing against a random player, the reason being that opponent's tendencies are later modeled individually and given as input to the neural network which should be able to convert this information into meaning.

Opponent modeling as described by Billings et al. [**?**, p. 208] tries to accumulate information about the playing style of an opponent. Later this information is applied to hand strength calculations and used to create likely probability distributions of opponent's hand ranges.

Again in our study opponent models are not exactly used in the same way as described but rather represent key opponent-specific tendencies which do not directly influence any calculations but rather serve as additional

input information for the neural network, to find an optimal betting strategy against different opponent types.

Unpredictability can be achieved by altering the strategy in a given situation. This means that a player should sometimes play differently in a similar situation, to not allow opponents to generate a precise model of the strategy [**?**].

This component is achieved in this thesis by following certain probability distributions when it comes to bet sizes and betting actions. A detailed description of the betting strategy follows in subsection **??**.

Bluffing is the last requirement mentioned by Billings et al. [**?**] and serves the purpose of sometimes winning pots with weak hands. By sometimes bluffing with weak hands we make opponents doubt our hand strength and later win more money on our strong hands against them.

The bluffing component was not explicitly design in our study, but ideally the neural network should discover the concept of bluffing on its own, by interpreting the given opponent models and its relative hand strength.

4.2 Betting Strategy

Billings *et al.* [**?**, p. 210] emphasis that betting strategies for pre-flop play and post-flop play are considerably different and that "a relatively simple expert

system is sufficient for competent play" in the pre-flop phase. Nonetheless for the conducted study no expert system for pre-flop betting decisions was used but an evolving neural network was provided with public game state information for all betting rounds. While sharing the same neural network for both pre-flop and post-flop play there is one significant difference in calculating the HS for pre-flop decisions compared to post-flop calculations.

4.2.1 Preflop Strategy

The state space in the pre-flop stage of a poker hand is relatively small compared to the post-flop state space. In total there are $\binom{52}{2} = 1352$ initial hole card combinations pre-flop, but this translates to only 169 distinct hand types because some hands have the same strength pre-flop but not post-flop. For example A♥ K♥ and A♦ K♦ have the same strength pre-flop and therefore are categorized into one distinct hand type, namely Ace King ***suited*** (AKs) [?].

For evaluating the hand strength of one agent against N opponents a lookup table was created. It consists of $169 * 8 = 1352$ entries, 169 distinct hand types played against one to eight opponents. A *Monte-Carlo Simulation* of one million poker hands was performed for each distinct hand type against each possible number of opponents. During those one million hands the agent's hole cards were ranked against all opponents ' hole cards and all

wins, ties and losses for the agent were counted. An approximation of the expected win percentage was calculated with following formula:

$$HS = \frac{(\#WINS + \frac{\#TIES}{2})}{\#WINS + \#TIES + \#LOSSES} \quad . \tag{4.4}$$

The hand strength (HS in the formula) represents the chance of a hand beating a random hand under the assumption of seeing all 5 community cards and going to showdown. This approach does not take opponent models into consideration [**?**].

Hand Ranker

A hand ranker as the name suggest, ranks a given poker hand based on its relative value by taking 5 to 7 cards as input and returning a number. The higher the number returned by the hand ranker, the better the hand [**?**]. There exists a variety of open source hand ranking algorithms on the internet but only a few of them were able to hold their own against the competition over the years. The reason why only a hand full of poker hand ranking algorithms are used nowadays is the need for speed. Writing a simple hand ranker is rather easy, but writing an efficient one that is able to rank millions of hands every second is not so trivial. Among the most famous hand ranking algorithms one especially excelled due to its speed. It was created by the *TwoPlusTwo (TPT)* community and uses a look up table with 32487834 entries which translates to a file size of approximately 250MB [**?**]. The TwoPlusTwo poker hand ranker is one of the fastest of its kind

to date. Not only is it extremely fast but also able to rank 5-card, 6-card and 7-card poker hands. This becomes very handy when evaluating hands pre-flop, on the flop and on the turn. For this reason the TPT hand ranker was used in multiple hand strength calculations throughout this study to determine if a given hand beats an other hand.

Hand Strength

While a hand ranker assigns a value to a given hand, it can only tell if a certain hand is better than other hands, and not how likely this hand currently is to win against a random hand. This is where the hand strength calculation comes into play. A simple approach of achieving this is to enumerate all possible remaining hand combinations an opponent could hold and count the number of wins, ties and losses versus the agent's hand. Using Formula **??** one gets the probability that the current hand beats a random opponent's hand at the current round of the poker game. This value is also called the *raw hand strength (RHS)*. While this approximation in combination with Monte-Carlo simulations might be sufficiently accurate for pre-flop hand strength assessment it is certainly insufficient for post-flop situations, when viewed in a vacuum. Raw hand strength calculations disregard the future potential of a hand and only represent the probability of a hand beating a random hand if there were no cards to come [**?**]. To overcome this obstacle the *Effective Hand Strength* algorithm was conceived by Billings et al. and first published in [**?**].

4.2.2 Postflop Strategy

For evaluating the strength of a hand after the pre-flop stage, it is important to not only consider the raw hand strength but also the positive and negative hand potential. Both metrics combined form the so-called *effective hand strength* [**?**].

Effective Hand Strength

The raw hand strength combined with the positive and negative hand potentials yield a measurement for the relative hand strength against an opponent competing in the poker hand. In general the effective hand strength can be represented by the following formula, as Billings et al. described it in [**?**, p. 216]:

$$EHS = HS \times (1 - NPot) + (1 - HS) \times PPot \quad , \tag{4.5}$$

where $NPot$ represents the negative pot potential, which stands for the probability of falling behind opponents when we currently have the best hand. Conversely $PPot$ stands for the positive hand potential, the probability of improving the hand when currently being behind the opponents. The EHS calculation can be generalized for n opponents by simply raising the HS to the power of *number of opponents* [**?**]. This yields:

$$EHS = HS^n \times (1 - NPot) + (1 - HS^n) \times PPot \quad . \tag{4.6}$$

Although Billings et al. do not recommend to generalize the EHS calculation we decided to do so in our study because the accumulated information

about tendencies of opponents was not directly used to influence hand strength calculations but rather serve as additional input to the neural network which in turn should learn to apply these models on its own.

Hand Potential

Hand potential calculations are used to account for future cards to come in a poker hand and to assess the possible impact of these cards on the current raw hand strength [**?**][1]. Assuming the current poker hand is in the flop-stage, with three cards already dealt to the community board, then there are $\binom{45}{2} = 990$ possible combinations of future turn and river cards for every possible hand an opponent might hold. That means if one would like to calculate the hand potential of a current hand against a random opponent that would equate to $\binom{47}{2} \times \binom{45}{2} = 1081 \times 990 = 1070190$ calculations for this situation.

Because such a number of calculations is too expensive for our training algorithm, the effective hand strength calculation was modified to approximate the EHS.

[1] More details on hand potential calculations and associated algorithms can be found in [**?**, p. 216-218].

Effective Hand Strength Approximation

Instead of calculating the *NPOT* and *PPOT* explicitly, the EHS approximation contains the *NPOT* and *PPOT* implicitly. A Monte-Carlos simulation similar to the one explained in Subsection **??** was used to determine a sufficient approximation of the EHS. Given an agent is currently one the Flop, Turn or River a simulation of 1000 games versus n active opponents is executed. If the agent is on the Flop, two cards are randomly drawn to simulate a possible Turn and River outcome, if he is on the Turn only one card is drawn and if he is on the River no additional cards are drawn. All n opponents, which have not yet folded their cards and are therefore still in the current poker hand, are assigned 2 random cards and then the number of ties, wins and losses of the agent vs all opponents are counted. This process is repeated a 1000 times before the counting results are evaluated with Formula **??**.

This simulation yields an approximated error of $\pm 4\%$ of the real hand strength at a given moment. This approximation is good enough for training the agents because an exact hand strength turned out to be not the single most important feature in order to develop a strategy.

4.2.3 Betting Strategy

When a neural network agent is prompted to execute an action he feeds the provided game information through his neural network and generates a *probability triple* at the output layer. The three output neurons represent the three possible moves (fold, check/call, bet/raise) an agent can execute. If an agents wants to *bet* or *raise* his hand he also needs to provide a betting amount in order for the move to be valid. This is realized by specifying two threshold values which decide the betting amount. Once an agent decides to bet or raise his hand, the value at the output of the third neuron (which corresponds to the *raise* action) is checked against the two thresholds (t_1 and t_2). An output value of less than t_1 results in a *small* bet amount, a value between t_1 and t_2 results in a *medium* bet amount and a value greater than t_2 results in a *large* bet amount.

In modern NL Texas Hold'em the main decider of a *small, medium* and *large* bet amount is the player's chip count (CC) [**?**]. In this thesis a *small* bet follows a normal distribution with a mean of $0.3 \cdot CC$ and a standard deviation of $0.05 \cdot CC$, while a *medium* bet follows a normal distribution with a mean of $0.45 \cdot CC$ and a standard deviation of $0.1 \cdot CC$. A *large* bet is normally distributed with a mean of $0.8 \cdot CC$ and a standard deviation of $0.05 \cdot CC$.

4.3 Training the NN-Agents

Unlike other games such as Chess, Checkers or Go, Poker is a game of imperfect information. This means, that a player can not see all relevant information at a given game state at all times. Because the game of Poker involves hidden information and deception, players must be willing to take risks based on the information they have at the current state of the game [**?**]. Unlike other methods, which reduce the large decision space of No-Limit Hold'em by transforming the game into a smaller abstract version in which they then approximate the optimal path and execute the result in the original game via a translation method, Nicolai and Hilderman [**?**] introduced a method, in which evolutionary algorithms can be used to train neural network agents to achieve reasonably good results in the game of No-Limit Texas Hold'em poker. The proposed algorithm in their work is used as a guide in this thesis to train neural network agents through iterative play over a large number of generations by mimicking natural evolution [**?**].

4.3.1 Evolution Phase

Evolutionary algorithms try to mimic the evolution process as it is know in biology. Agents in a population are given a task and try to achieve the best results possible. The performance of agents is then evaluated and the best among the population are chosen to reproduce offsprings for the next

generation. Offsprings are almost identical to their parents, but with slight changes to their genetic material. These offsprings together with the best agents from the previous generation are then again given the same task and the whole process repeats. [**?**, **?**].

In case of our neural network agents, the population of the first generation consists of randomly weighted neural network agents. They are competing in a number of tournaments against each other. The best agents of a generation are then chosen to reproduce offsprings. Together with the best agents, all offsprings are then competing in another series of tournaments until after N generations the process is stopped.

Figure **??** shows the implementation of simulating a number of tournaments for each generation and selecting a number of elite players per generation to evolve the neural network agents.

This algorithm shows the procedure of selecting the fittest agents of each generation and creating offsprings by evolving these elite players. The algorithm takes as input a vector of *players*, which represents all the randomly created initial neural network agents, another vector called *hall_of_fame*, which is initially also filled with randomly created neural network agents and yet another vector called *population*, which forms the initial playing population consisting of all players and hall of fame members. Line 10 initialized the rules of the game, including the blind structure, the number of players per table, the starting chip stack for each player and many more properties. Line 12 declares a vector of elite players which is filled in Line 13 after *NUM_TOURNAMENTS_PER_GEN* tournaments are played in the current generation. After all tournaments have concluded the *elite* vector holds the *NUM_ELITE_PLAYERS* best players of the current generation. The best players of a given generation are determined by a fitness function explained in Subsection **??**. In Line 15 the hall of fame is updated by a procedure which will be described in the following subsection **??**. In Line 18 the vector of elite players is then passed to Figure **??**.

The *evolvePlayers* function takes as input a vector of *players*, which represents all agents of the current generation ordered by their overall fitness, and a vector called *elite_players* which holds the fittest players of the current generation. This function handles the creation of new neural network agents by evolving pre-selected elite players of a given generation.

Lines 7 through 37 loop through all non-elite agents of the current generation which will later represent the newly created agents. Offsprings can have

```
define NUM_ELITE_PLAYERS 5;
define NUM_TOURNAMENTS_PER_GEN 200;
define MUTATION_LIKELIHOOD 0.10;
define NUM_GENERATIONS 2000

void evolvePlayers(std::vector<Player*> players, std::vector<Player*> hall_of_fame
    , std::vector<Player*> population) {
    for (int i = 0; i < NUM_GENERATIONS; i++) {

        /// EVALUATION PHASE
        Rules* rules = defineRules();

        std::vector<Player*> elite;
        elite = runOneGeneration(population, players, rules, NUM_ELITE_PLAYERS,
            NUM_TOURNAMENTS_PER_GEN, hall_of_fame);

        updateHallOfFame(population, elite_players, hall_of_fame);

        /// EVOLUTION PHASE
        evolvePlayers(players, elite, MUTATION_LIKELIHOOD);
    }
}
```

Figure 4.3: The evolution process to train poker agents.

```
void evolvePlayers(std::vector<Player*> players, const std::vector<Player*>
    elite_players) {

    std::discrete_distribution<int> custom_distribution {4,25,6,1,1,1,1};
    std::uniform_real_distribution<> uniform_real_distribution(0,1);
    std::normal_distribution<double> normal_distribution(0, 0.6);

    for (long p = elite_players.size(); p < players.size(); p++) {

        int parent_count = custom_distribution() + 1;

        if (parent_count > elite_players.size()) {
            parent_count = elite_players.size();
        }

        std::vector<Player *> parents = chooseParents(parent_count);
        std::vector<double> parents_evolution_weights = assignParentsWeights(
            parent_count);

        std::vector<double> child_weights = players(p)->getOutputWeights();

        for (int w = 0; w < child_weights.size(); w++) {
            double new_weight = 0.0;

            for (int par = 0; par < parents.size(); par++) {
                std::vector<double> parent_weights = parents.at(
                        par)->getOutputWeights();
                new_weight += parent_weights[w] * parents_evolution_weights[par];
            }

            double random_value = uniform_real_distribution();
            if (random_value < MUTATION_LIKELIHOOD) {
                double noise_value = normal_distribution();
                new_weight += noiseValue;
            }
            child_weights.at(w) = new_weight;
        }
       players.at(p)->setOutputWeights(child_weights);
    }
}
```


Figure 4.4: The creation of offsprings by evolving elite players of a generation.

zero to *NUM_ELITE_PLAYERS* parents. The number of parents for each new child is determined in Line 9. The most likely parent count is *two*, followed by *three* and *one*, but there is also a slight chance that the number of parents is larger than three.

Once the number of parents is determined, the parents for reproduction are chosen in Line 15. Each *elite_player* is equally likely to be chosen but can only be chosen once. Line 16 determines the influence of each parent on the reproduction. Each parent is assigned a weight between zero and one, while the sum of all weights equals one. Next all connection weights of the neural network of a child are loaded into a vector in Line 18. Lines 20 through 35 loop over all weights of the child and update them. The updated weight of each neural network connection is a biased average of the weights of the neural network of the parents [**?**]. This is represented by the Lines 23 through 27.

To avoid getting easily stuck in local minima and to explore the decision space, mutation is applied to the connection weights with a certain *MUTATION_LIKELIHOOD*. The mutation is represented by adding normally distributed noise with a mean of 0 and a standard deviation of 0.6 to the new weight of the child. The last step, shown in Line 36, is to apply the newly created weights to the child.

4.3.2 Fitness Function

A mixed fitness function, consisting of three components is used in this thesis to evaluate the performance of evolved neural network agents. The choice of the three components was greatly influenced by the work of Nicolai in [**?**]. The first component is the *average ranking* in a tournament and can be obtained by ranking agents after each tournament, accumulating the results and then averaging them after a set of tournaments was played. The second component represents the *average number of hands won per tournament*. The third component of the mixed fitness function represents the *mean money won per tournament*. The mean money won can be calculated with Equation **??**.

$$MeanMoney = \frac{MoneyWon}{HandsWon} - \frac{MoneyLost}{HandsLost} \tag{4.7}$$

MoneyWon represents the amount of chips a player has won over a set of tournaments, likewise *MoneyLost* represents the amount of chips a player has lost. *HandsWon* and *HandsLost* are the number of hands a player has won and lost in this set of tournaments, respectively [**?**]. While this fitness function is used to evaluate and train agents in a given generation, another fitness function is later used to further benchmark the performance of the best agents per generation. The mixed fitness function is designed in a way that it rewards aggressive play more than passive play, because in real life poker aggressive strategies have shown to be superior to passive strategies. However just based on the average ranking in a tournament one can not

conclude that a lower average ranking also means that the strategy is more profitable. Profitability in a poker tournament is measured by the so-called *Return of Investment (ROI)*. To calculate the ROI of poker tournaments the accumulated winnings of all tournaments minus the accumulated cost of all tournaments is divided by the cost of all tournaments [**?**].

$$ROI_{\%} = \frac{Acc.Winnings - Acc.Cost}{Acc.Cost} \cdot 100 \tag{4.8}$$

The overall fitness of an agent can be calculated as a weighted sum of all three components:

$$OverallFitness = w_1 \times Ranking_{avg} + w_2 \times HandsWon_{avg} + w_3 \times MeanMoney \tag{4.9}$$

All three components are first linearly mapped to a range between 0 and 1 so that the weights w_1, w_2 and w_3 represent a percentage importance. Furthermore while an increase of *HandsWon* and *MeanMoney* indicate an improvement, it does not for the *Ranking* component. Therefore the linearly mapping was reversed for this component.

4.3.3 Hall of Fame

In [**?**, p. 63] Nicolai explains the non-transitive nature of poker. While agents of generation *A* might defeat agents from generation *B* frequently and agents from *B* regularly defeat agents from generation *C*, it doesn't automatically follow that agents from *A* also defeat agents from *C* frequently.

This means that there is no guarantee that an evolutionary system improves with each generation.

To counter this inherent problem with evolutionary algorithms a measure called *Hall of Fame (HOF)* is implemented in this thesis. Successful strategies of previous generations are stored in the HOF and serve as a benchmark for newly created generations [**?**]. Together with players of the current generation, hall of fame members compete in a series of tournaments. All players are then ranked according to their overall fitness but members of the hall of fame and agents of the current generation are sorted individually. This means that all members of the HOF are sorted by their relative ranking in this generation and likewise agents of the current generation which are not members of the HOF are sorted relatively to their overall fitness.

The worst #x players of the HOF are then replaced by the best #x players of the current generation. In Figure **??** an exemplary representation of the individual sorting of HOF members and agents of the current generation is shown. In this example the number of possible replacements is chosen to be two. That means that the two best agents of the current generation replace the two worst members of the HOF because their overall fitness is better. If the overall fitness of the two best agents of the current generation would have been worse than the overall fitness of the worst HOF member, no replacement would have occurred. The new HOF is then used to serve as a benchmark for the next generation of newly evolved agents. HOF members are not involved in the evolution process, only agents of the current generation are.

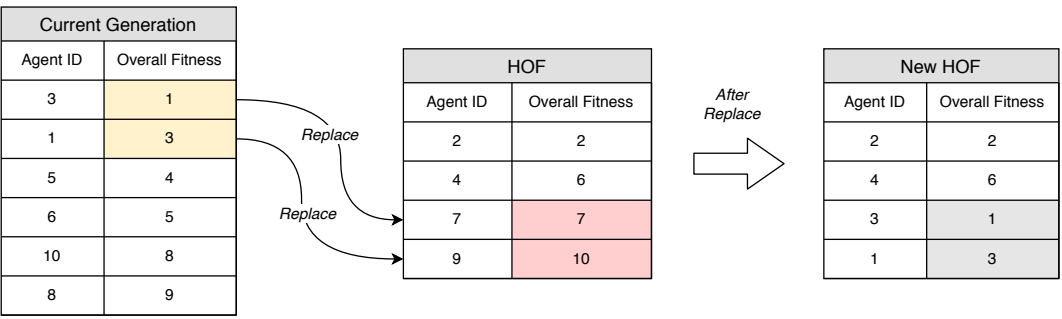

Figure 4.5: Example of Hall of Fame member replacement.

5 Experimental Results

In this chapter, the results of three different methods for training an evolutionary neural network poker agent are presented. In the first experiment a baseline control agent was created using an evolutionary neural network without any countermeasures accounting for problems with evolutionary algorithms. In the second and third experiment a hall of fame was introduced to keep strategies, which have proven to be strong in previous generations, in the playing population. Additionally, in the third experiment playing tendencies of opponents were modeled and given as input to the neural network, with the goal to further improve the ability to adjust to certain playing styles.

The skill of the best agent after a number of generations was then evaluated with two different metrics against a set of static opponents for each experiment.

5.1 Benchmark Opponents

To test the skill of evolved neural network agents, they played in a number of tournaments against predefined static opponents. In a series of tournaments all players are ranked by the fitness function described in Subsection x.x. For all benchmark tests the same weight distribution for the fitness function was used to make the results of different evolution methods comparable to each other. While the *average placement* in a tournament might be a strong indicator for the level of skill of a poker player, it certainly should not be used as a benchmark value on its own. In this thesis a combination of three benchmark values was used to assess the skill of an agent. Furthermore another fitness function is later used to determine the profitability of a poker agent by calculating the *return of investment* over a series of tournaments. This fitness function is often used in real live poker because success of tournament players is measured by the amount of money they won in their poker career.

5.1.1 Always Fold

An *Always Fold* agent does exactly what his name suggests, he always folds his two hole cards when it is his turn to bet. The only exception to this rule is when the action allows to check instead of folding. A folding strategy is very effective in a static poker environment, where agents follow static rules and do not exploit weaknesses of their opponents. While a folding strategy

can never win against a betting strategy in tournaments, it might frequently reach high ranks in tournaments. This is because a folding strategy can never lose more chips than the *big blind* in a single hand, while more aggressive strategies frequently bust out of a tournament earlier due to the active betting against opponents.

5.1.2 Always Call

An *Always Call* strategy calls any bet made at the table at any time. If however there is the chance to check, it will do so.

5.1.3 Always Raise

Always Raise agents on the other hand raise a previous bet whenever there is the chance to do so or bet themselves if no previous bet was made yet.

5.1.4 Random

The *Random* agents implemented in this thesis have a 25% chance of folding or check a hands, 30% chance of raising with a hand, 44% chance of calling with a hand and a 1% chance of going all-in with a hand. The percentage values for each action were arbitrarily chosen to represent a pseudo random behavior.

5.2 Evolution without HOF

In the first conducted experiment 45 randomly created neural network agents played for 200 tournaments per generation. A network topology of $16 - 12 - 3$ was used for all participating agents. After each generation agents were ranked according to Formula **??**, with a weight distribution of $w_1 = 0.8, w_2 = 0.02, w_3 = 0.18$. As indicated by the weights the main focus for agents was to achieve a low *average ranking*, while at the same time maximize the *mean money* won in tournaments. The weight for the *hands won* component of the overall fitness function was considered as being not so important for overall success in tournaments. The stated weights for training the neural network agents were established by trial and error and the most fitting set was chosen for all experiments.

The best performing 10% of agents were then selected as possible parents for the evolution phase. This equated to 5 possible parents for 40 newly created offsprings. This new population consisting of the 5 best agents of the previous generation and the 40 new agents then played for another 200 tournaments. This process was repeated for 1000 generations after which the best performing agent was selected to represent a *baseline Control agent*.

5.2.1 Skill Progression

Figure **??** shows a *moving average* of the progression of skill in the playing population over all 1000 generations. The vertical axis corresponds to the overall fitness of the best performing agent in a given generation. The subset size for the moving average was set to 50 data samples, which yields a smooth representation of the fitness progression.

A more detailed view of the individual components of the overall fitness function can be seen in Figure **??**, where the fitness is split up into all three components. Each factor is represented as a moving average of size 50 over all generations.

somewhere say that the results were created by simulating 1000 tournaments. The performance of the *Control* agent was then evaluated against 44 evenly distributed static opponents. To establish a baseline ranking for all player types, the population consisting of 11 agents for each static opponent type (*Always Fold, Always Call, Always Raise, Random*) and one *Control* agent competed in 100,000 tournaments and were again ranked by their overall fitness as described in Subsection **??**. The results of this baseline evaluation are shown in the histogram in Figure **??**.

Figure **??** clearly indicates an overall skill progression over generations. The non-steady increase in skill can be explained by the non-transitive nature of poker, as described in Subsection **??**, and by an inherent problem of evolutionary algorithms known as *Evolutionary Forgetting*. In short this

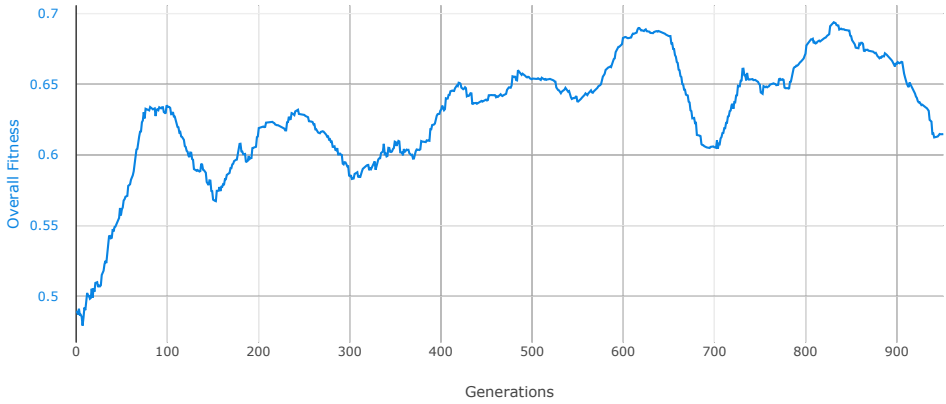

Figure 5.1: Overall fitness of *Control* agent over 1000 generations.

phenomenon can be defined as "the tendency for a population to lose good strategies as they are replaced by seemingly better ones." [**?**, p.63]. This means that the overall skill level of a generation might decrease due to loosing some strategies, which have proven to be strong. Nonetheless the baseline *Control* agent did beat his competition over 100,000 tournaments, as shown in Figure **??**.

The best overall fitness of 0.6578 was achieved by the Control agent, closely followed by the static *Always Fold* opponents.It should be mentioned that the results of the static opponents were averaged. The *Always Raise* agents performed poorest, with a fitness score of only 0.2467. *Always Call* and *Random* agents also did not perform well, receiving a fitness score of only 0.3124 and 0.3595, respectively. One could think, based on this results, that a

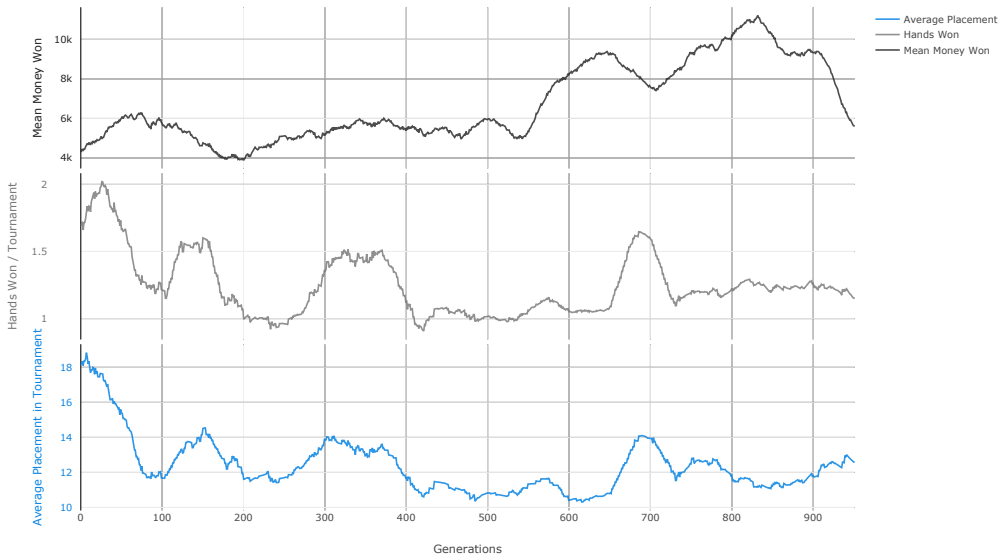

Figure 5.2: Evolutionary progress for all three individual fitness components over 1000 generations

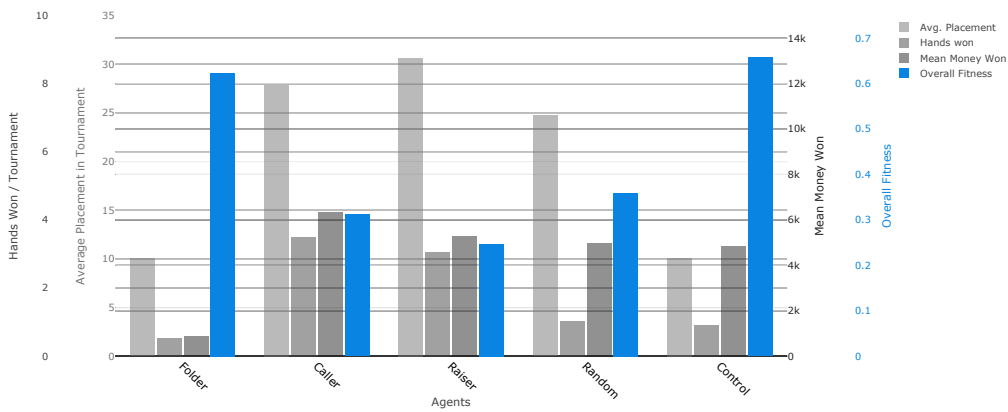

Figure 5.3: Baseline fitness of all playing styles including the *Control* agent.

folding strategy might be a good choice, which indeed holds true in general, however an *Always Fold* agent will never win a tournament but at best reach second place [**?**].

Alternative Fitness Evaluation

In addition to the *weighted overall fitness* function, which was also used for training the agents, another independent fitness function was evaluated to compare the overall profitability of playing styles. In Figure **??** the vertical axis corresponds to the percentage *ROI* for each of the playing styles listed on the horizontal axis.

To calculate the ROI of all agents, the tournament buy-in was set to 1$ and a progressive ***payout structure*** shown in Table **??** was used to reward the seven best ranked players. While *Always Fold* agents achieved an almost equal *overall fitness*, the *Control* agent won far more money over 100,000 tournaments than the second best competitor. With a ROI of 80.27% the *Control* agent's strategy looks far superior compared to a ROI of only 20.52% for the *Always Fold* strategy. However a ROI greater than zero indicates a profitable strategy, which means that only the *Control*, the *Always Fold* and *Always Call* agents were profitable players in the conducted experiment. With a negative ROI of -73.31% the *Random* strategy performed worst, followed by an almost break even ROI of -0.9% for the *Always Raise* agent. Figure **??** shows the profitability of the *Control* agent over 1000 generations. The data was smoothed by a moving average with a subset size of 50. As

Rank in tournament	Price money (% of buy-in)
1	31 %
2	21.5 %
3	16.5 %
4	12.5 %
5	9 %
6	6 %
7	3.5 %

Table 5.1: Progressive payout structure for a tournament with 45 players.

discussed earlier this graph again shows a non-steady progression of skill over generations. A lot of ups and downs indicate, that good strategies were lost in the phase of evolution. In the end however an overall increase in performance over the span of 1000 generations can be observed. Both fitness evaluations therefore have shown, that the skill of the *Control* agent did progress in this experiment.

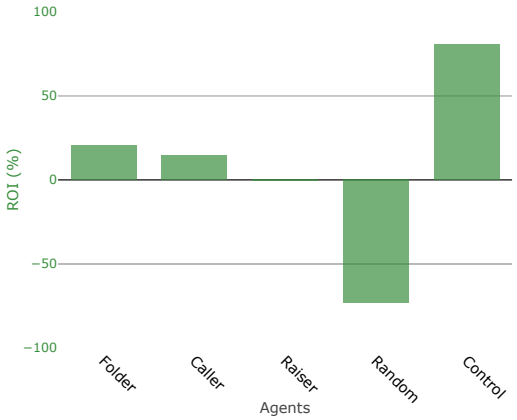

Figure 5.4: Profitability of different playing styles measured by the percentage *ROI*.

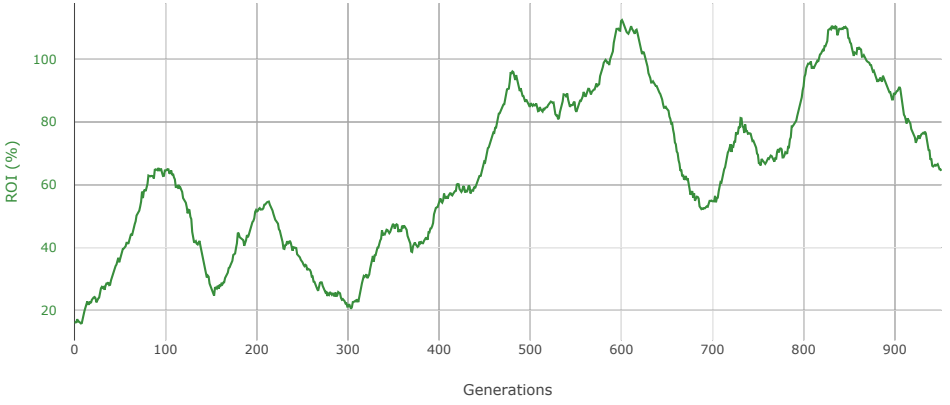

Figure 5.5: Mean dollar won over 1000 generations.

5.3 Evolution with HOF

Is the network topology needed here? Or should be talk about the network topology in the introduction paragraph? In this experiment the effect of introducing a hall of fame to the playing population was evaluated. Once again a starting population of 45 randomly created neural network agents played in 200 tournaments in the first generation. This time however the playing population was split into two groups. The first group was represented by the members of the hall of fame (later referred to as *HOF-group*), which were not part of the evolution process, and the second group was formed by the remaining agents, which were possible candidates for reproduction (later referred to as *evolution group*). In this experiment the *HOF-group* consisted of 13 agents, while the *evolution group* consisted of the remaining 32 agents. After the completion of the last tournament of each generation a procedure, which is explained in detail in Subsection **??**, ranked all agents according to their fitness, updated the hall of fame and selected the best performing 10% of agents as parents for the evolution phase. Newly created agents together with all current hall of fame members then competed in another 200 tournaments. After 1000 generations the best performing agent of the *evolution group* was chosen to represent the *HOF Control* agent.

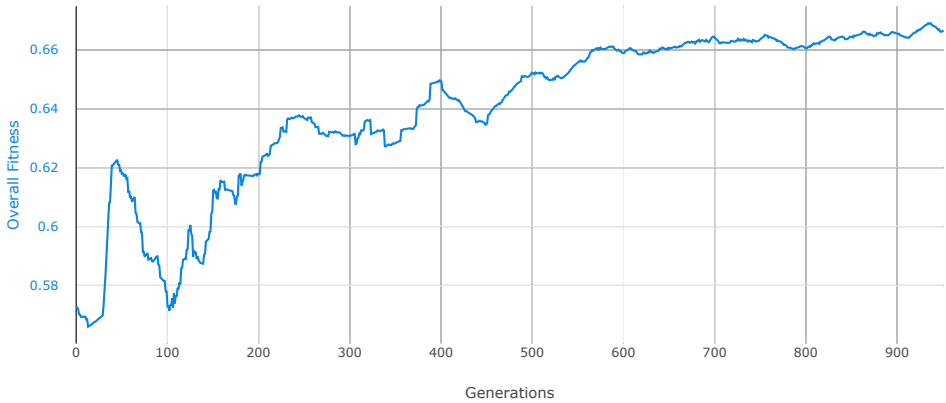

Figure 5.6: Overall fitness of *Control* agent over 1000 generations.

5.3.1 Skill Progression

Similar to the first experiment Figure **??** shows the progression of skill in the *evolution group*. All data sets in this experiment were smoothed by a *moving average* of size 50 — maybe put this in the introduction and say it was used for all experiments?. The individual components of the overall fitness function can be seen in Figure **??**. The performance of the *HOF Control* agent was measured over a set of 100,000 tournaments against the same benchmark agents, which were used in the first experiment. Figure **??** shows the final results for each group of benchmark agents and for the *HOF Control* agent.

The curve of the overall fitness of the *HOF Control* agent shows much less variance over the generations compared to the overall fitness of the *baseline*

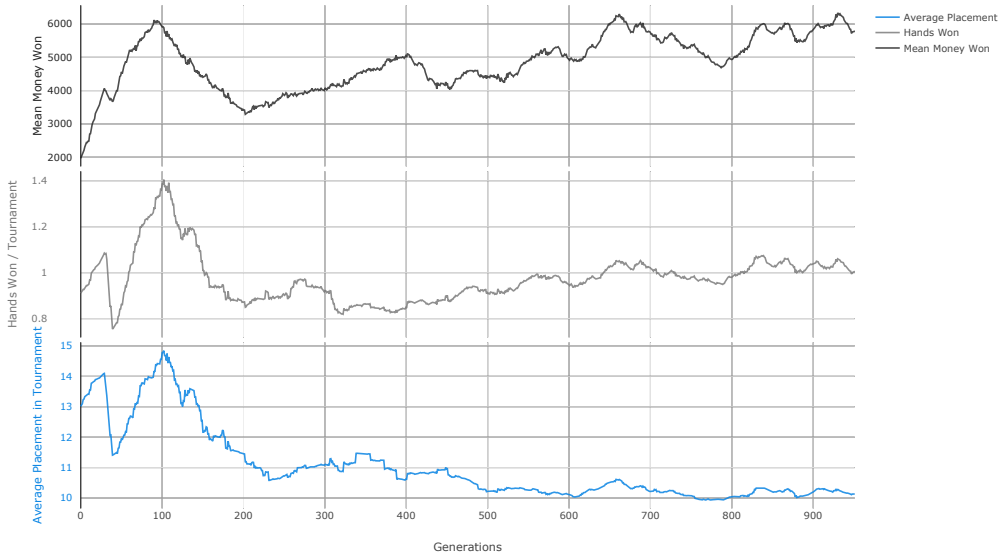

Figure 5.7: Evolutionary progress for all three individual fitness components over 1000 generations

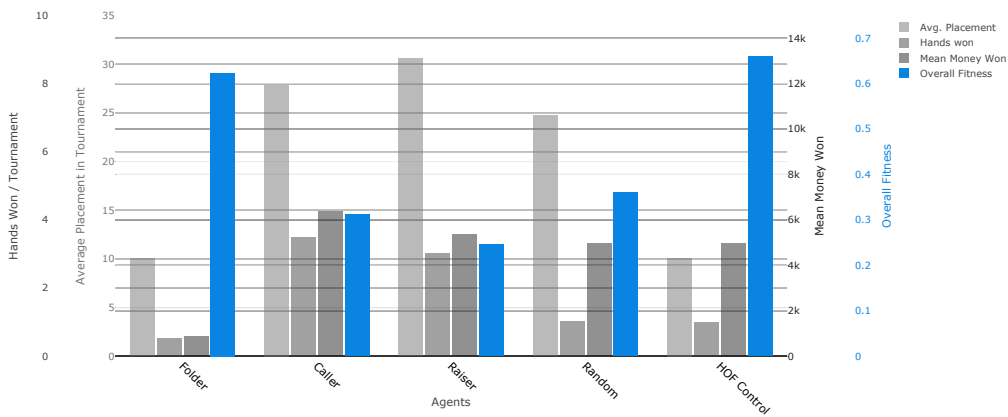

Figure 5.8: Baseline fitness of all playing styles including the *Control* agent.

Control agent. Also the fitness trend is much more recognizable in the hall of fame experiment. Figure **??** furthermore confirms this observation and shows a much more steady trend of all fitness components. After generation 200 a steady trend is present for all three individual components of the fitness function, and the overall fitness therefore looks much smoother compared to the first experiment. The reason for a less deviant progression in skill lies in the use of a hall of fame. By keeping strategies in the playing population, which have proven to be able to beat the majority of the current population, it is less likely that a seemingly good strategy is chosen for reproduction. Only strategies that can beat the current *evolution group* and also members of the hall of fame are chosen for reproduction. This yields a much more steady progression in skill but not necessarily a faster one. The results shown in Figure **??** indicate that the fitness of the *HOF Control* agent is only slightly better than the fitness of the *baseline Control* agent after 1000 generations. The overall fitness of the *HOF Control* agent improved by only 1% from 0.6578 to 0.6603, the overall fitness of the benchmark agents also did not change significantly.

The biggest difference between the *HOF Control* agent and the *baseline Control* agent is the profitability of their playing styles. In Figure **??** a clear increase in the percentage ROI can be observed. The *HOF Control* agent achieved a respectable 93.35 % ROI over 100,000 tournaments, which is an increase of 16.3% over the *baseline Control* agent, while all other benchmark agents experienced a deterioration in their ROI.

Figure **??** once again proves the point, that the agents evolved in the hall of

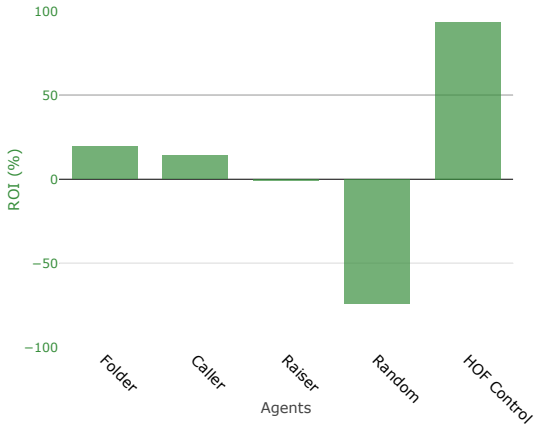

Figure 5.9: Profitability of different playing styles measured by the percentage *ROI*.

fame experiment show a much more stable improvement over generations. Even though the individual results of the best agent after 1000 generations is very similar to the results of the *baseline Control* agent, there are a lot of indications that the *HOF Control* agent is a much stronger opponent over increasing generations. Maybe mention something else here, that supports this claim?.

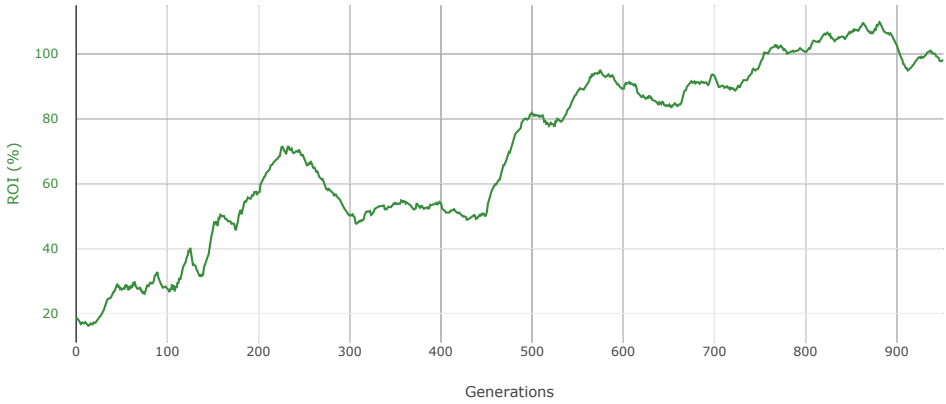

Figure 5.10: Mean dollar won over 1000 generations.

5.4 Evolution with HOF & Opponent Modeling

The objective of the third and final experiment is to investigate the influence of opponent modeling on the results of the playing population described in the second experiment. While agents of the first and second experiment had a neural network topology of $16 - 12 - 3$, agents in this experiment were trained with a network topology of $40 - 22 - 3$. The additional 24 inputs to the neural network are used to model playing tendencies of opponents, as it is described in detail in Subsection **??**. The number of neurons for the hidden layer of the neural network was chosen arbitrarily and turned out to be a suitable amount for the conducted experiment.

In this experiment the playing population was again split into two groups.

As described in the second experiment the *HOF-group* and the *evolution group*, consisting of 13 players and 32 players, respectively, competed in 200 tournaments over 1000 generations. The procedure of ranking players and updating the hall of fame stayed the same in this experiment. After the completion of the last tournament in the 1000th generation the best performing agent of the *evolution group* was chosen to represent the *HOF_OPM Control* agent. *HOF_OPM* in this case stands for *Hall of fame with opponent modeling*.

5.4.1 Skill Progression

Figure **??** shows the overall fitness of the *evolution group* over 1000 generations and Figure **??** shows the progression of the individual fitness components over time. To test the performance of the *HOF_OPM Control* agent a set of 100,000 tournaments was played against the same benchmark agents from the first two experiments. The results of this evaluation can be seen in the histogram in Figure **??**.

The overall fitness curve in Figure **??** shows even less variance over the generation than it did in the second experiment. While the fitness curve in Figure **??** shows an upward trend over 1000 generations, the fitness curve in this experiment starts flattening out from generation 300 onwards. The same effect can be seen in Figure **??**, where all three fitness components do not change very much from generation 300 onwards. This might indicate that the neural network got stuck in a local minimum and was not able

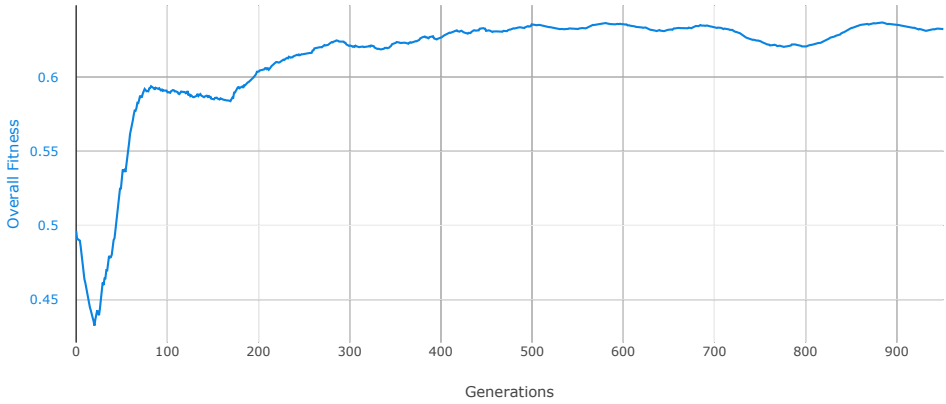

Figure 5.11: Overall fitness of *Control* agent over 1000 generations.

to escape from it due to too little mutation applied in the evolution phase. While this is one possible explanation to the observed effect, another reason could be a non optimal number of neurons for the hidden layer. A set of different network topologies was tested throughout this experiment, but all variations yielded similar results. Compared to both other control agents from the first and second experiment, the profitability of the *HOF_OPM Control* agent, seen in Figure **??**, shows a fast rise between generation 150 and 400, but levels off from there on. While the averaged *ROI* never exceeds the 90% mark, it also does not show much variance, which is a good sign and might indicate that good strategies are not as easily lost as in the other experiments. With a *ROI* of 82.94% against the static competition, the *HOF_OPM Control* agent performed worse than the *HOF Control* agent but better than the *baseline Control* agent. All static agents performed similar

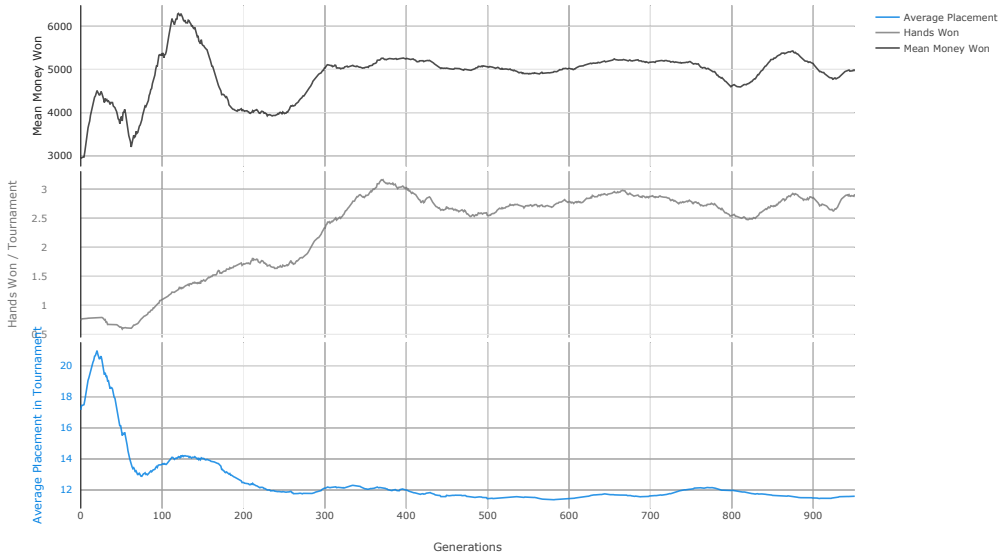

Figure 5.12: Evolutionary progress for all three individual fitness components over 1000 generations

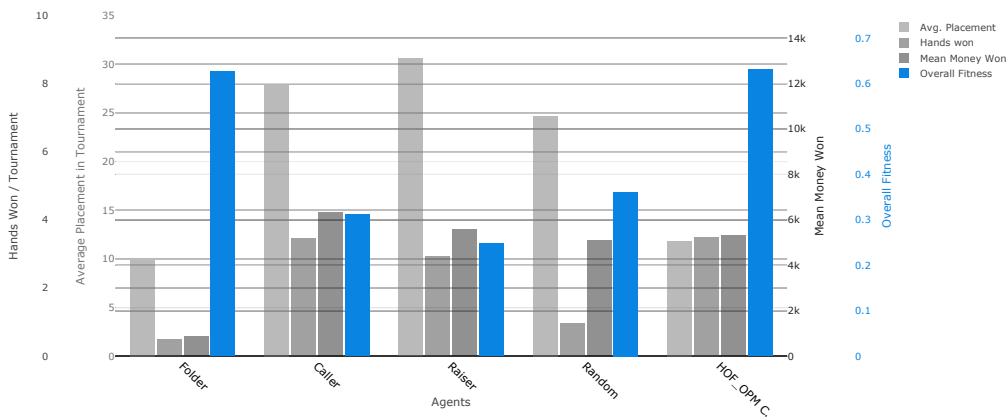

Figure 5.13: Baseline fitness of all playing styles including the *Control* agent.

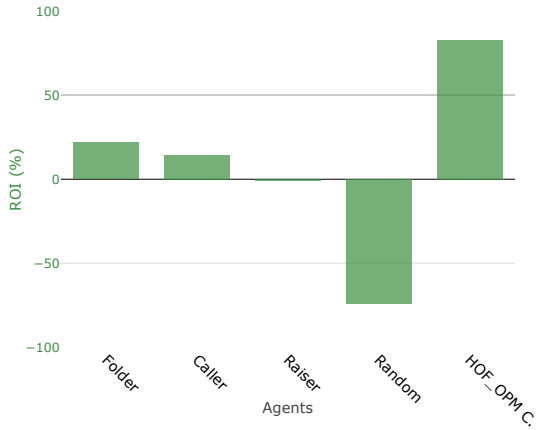

Figure 5.14: Profitability of different playing styles measured by the percentage *ROI*.

to the first two experiments, reaching *ROI* values of 21.95% (*Always Fold* agent), 14.34% (*Always Call* agent), -1.01% (*Always Raise* agent) and -74.02% (*Random* agent), shown in Figure **??**.

To better understand the obtained results, the playing styles of all three neural network populations in the three different experiments were compared to each other.

5.4.2 Playing Style Development

Figures **??**, **??** and **??** show the playing style progression of the trained population for the first, second and third experiment, respectively. The graph consists of three statistical values used to describe the playing style of

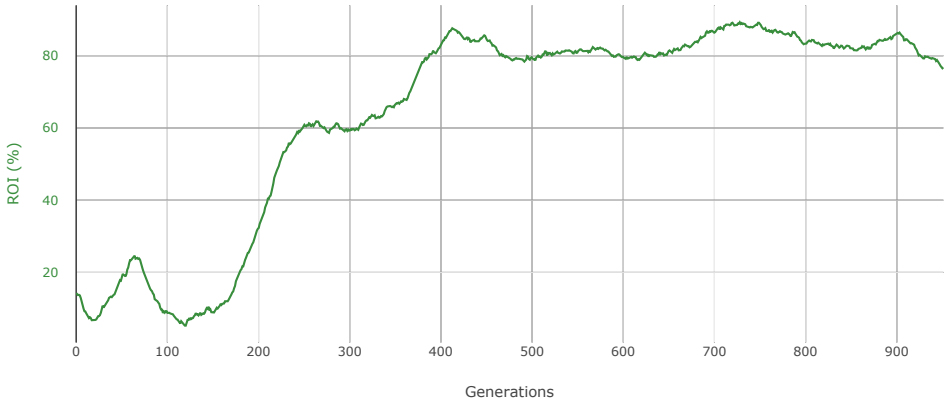

Figure 5.15: Mean dollar won over 1000 generations.

a poker player. The three vertical axes correspond to the *VPIP*, *PFR* and *AFq*, three basic statistical values used to describe tendencies of poker players, which are explained in detail in Subsection **??** under *Opponent model for all opponents*.

The *VPIP* combined with the *PFR* gives a good estimation of how opponents are playing and how they can be exploited. The *VPIP* value of a player will always be higher than the *PFR* value, this is because the *PFR* is increased with every raise action before the flop, while the *VPIP* value is increased with every call or raise action before the flop. In general a higher *PFR* value indicates a more aggressive playing style, and a big gap between the *VPIP* value and the *PFR* value indicates a more passive style. But there is more room of interpretation when comparing these two values [**?**].

A very high *VPIP* paired with a low *PFR* (i.e vpip: 70, pfr: 6) indicates a very weak player who plays too many hands pre-flop and only raises, when he has a very strong hand. Having this type of player on the table can be very profitable, because they usually play very passive and give away money too easily. A low *VPIP* paired with a low *PFR* (i.e vpip: 10, pfr: 8) on the other hand indicates a very tight range of hands and means that these players usually play very aggressive when they opt to play. Very good players have a small gap between *VPIP* and *PFR*. Their values range between 15 to 28 for the *VPIP* and 14 to 23 for the *PFR*. For example a good player might have a *VPIP* of 22 and a *PFR* of 18 but could easily also have values like (vpip: 19, pfr:17) or (vpip: 25, pfr: 23) [**?**].

Figure **??** shows that agents in the first experiment, where no hall of fame members competed in tournaments, started off with a high *VPIP* value ranging from 34 to almost 40 and a low *PFR* value of around 10. After 100 generations however, the gap between these two values started to shrink, which is a clear sign of improvement. Agents in the second experiment showed the same behavior but succeeded a little bit early in recognizing the importance of keeping the *VPIP* value close to the *PFR* value. This can be seen in Figure **??**. In both experiments agents kept a small gap between these two values throughout the rest of all generations. The main difference between this two experiments is that agents competing against hall of fame members performed consistently well with a very low *VPIP* and *PFR* in the range of 2 to 3, while agents without a hall of fame competition frequently switched between *VPIP* and *PFR* values in the range of 5 to 25.

A big difference in an agent's playing style can be observed in Figure **??**, which shows the results for the third experiment. The introduction of an opponent model as input to the neural network led to a very different playing style progression over generations. Similar to the first two experiments, agents in this third population also recognized the importance of a small gap between *VPIP* and *PFR*, at least until generation 70, from where on the *VPIP* value starts to diverge from the *PFR* value significantly. From generation 500 onwards the *VPIP* value started to level off at a value of around 22, while the *PFR* value stayed low at around 2. The playing style associated with these values indicates a very tight player, who patiently waits for the best hands and only raises with them. However this kind of player might also trap other players with some tricky plays or might develop a call/re-raise strategy [**?**]. While this kind of playing style is not considered to be the best, it is by far not the worst strategy. When playing against very static opponents, this strategy might even be advantageous over other tight strategies.

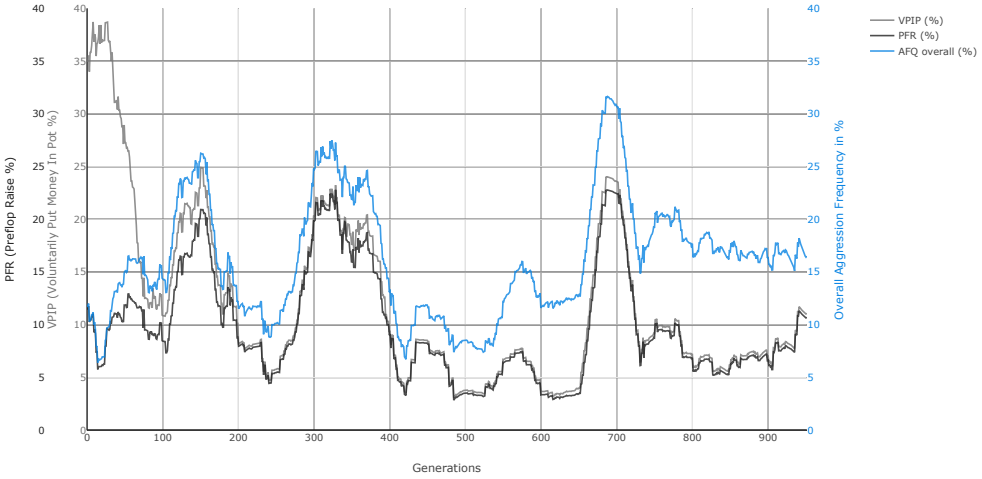

Figure 5.16: Player statistics (VPIP, PFR, AFq) over 1000 generations.

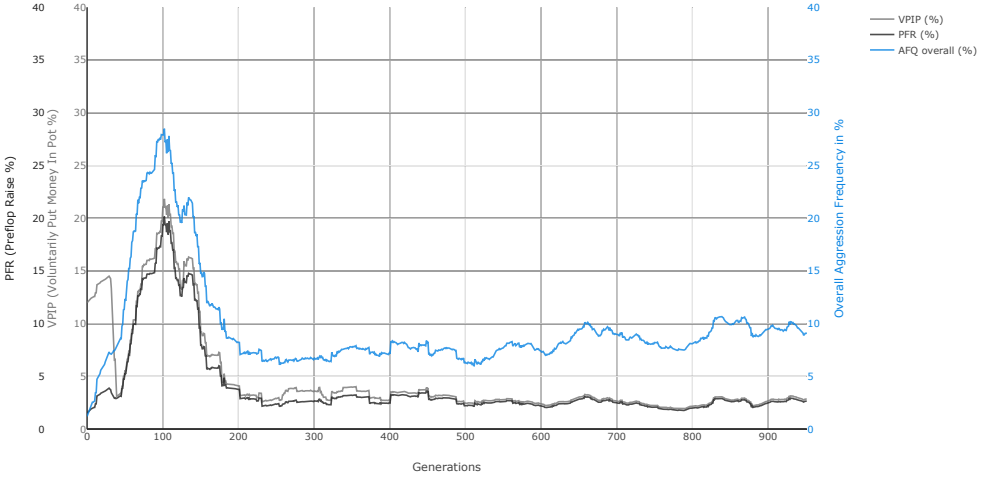

Figure 5.17: Player statistics (VPIP, PFR, AFq) over 1000 generations.

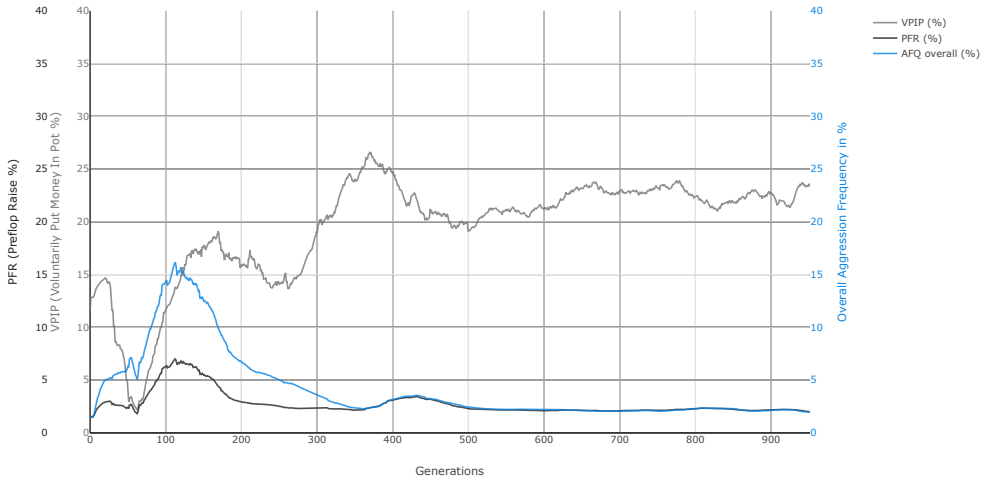

Figure 5.18: Player statistics (VPIP, PFR, AFq) over 1000 generations.

6 Conclusion and Future Work

This is a placeholder for the conclusion and future work.

Bibliography

[1] D. Billings, A. Davidson, J. Schaeffer, and D. Szafron. *The Challenge of Poker* Artificial Intelligence 134, pp. 201-240, 2002.

[2] L. Peña. *Probabilities and simulations in poker.* Mather's thesis, Department of Computing Science, University of Alberta, 1999.

[3] N. Passos. *Poker Learner: Reinforcement Learning Applied to Texas Hold'em Poker*, Master's thesis, Faculdade de Engenharia da Universidade do Porto, Portugal, 2011. https://paginas.fe.up.pt/~eio8029/Master Thesis - Nuno Passos.pdf

[4] J. Schaeffer, D. Billings, L. Peña, and D. Szafron. *Learning to Play Strong Poker*. ICML-99, Proceedings of the 16th International Conference on Machine Learning, 1999. http://poker.cs.ualberta.ca/publications/ICML99.pdf

[5] J. Rubin, and I. Watson. *Computer poker: A review.* Artificial Intelligence, 175(5-6):958-987, 2011.

[6] D. Sklansky. *The Theory of Poker.* Two Plus Two Publishing, 2005.

[7] D. Billings. *Algorithms and Assessment in Computer Poker* Ph.D. Dissertation, University of Alberta, 2006.

[8] R. D. Harroch, and L. Krieger. *Poker For Dummies.* John Wiley & Sons, 1st Edition, 2000.

[9] M. J. Osborne. *A Course in Game Theory.* The MIT Press, Cambridge, Massachusetts, London, England, 2014.

[10] V. Lisy, and M. Bowling. *Equilibrium Approximation Quality of Current No-Limit Poker Bots.* arXiv preprint arXiv:1612.07547, 2017.

[11] D. Billings, L. Peñe, J. Schaeffer, D. Szafron. *Using Probabilistic Knowledge and Simulation to Play Poker.* In 16th National Conference on Artificial Intelligence, pp. 697-703, 1999.

[12] D. Billings, D. Papp, J. Schaeffer, and D. Szafron. *Opponent Modeling in Poker.* Proc. AAAI-98, Madison, WI, pp. 493-499, 1998.

[13] A. Davidson. *Opponent modeling in poker: Learning and acting in a hostile and uncertain environment.* Master's thesis, University of Alberta, 2002.

[14] G. Nicolai, and R. J. Hilderman. *No-Limit Texas Hold'em Poker Agents Created with Evolutionary Neural Networks.* CIG-2009, IEEE Symposium on Computational Intelligence and Games, pp. 125-131, 2009.

[15] G. Nicolai. *Evolutionary methods for learning no-limit Texas hold'em poker.* Master's thesis, University of Regina, 2008.

[16] L. F. Teofilo. *Estimating the Probability of Winning for Texas Hold'em Poker Agents*. Proceedings 6th Doctoral Symposium on Informatics Engineering, pp 129-140. Porto, Portugal.

[17] G. Nicolai, and R. J. Hilderman. *Algorithms for Evolving No-Limit Texas Hold?em Poker Playing Agents.* In Proceedings of the International Conference on Evolutionary Computation, 2010.

[18] *Chapter 1: VPIP and PFR* [Internet]. Poker Copilot. 2016 [cited 23 October 2018]. Available from: https://pokercopilot.com/poker-statistics/vpip-pfr

[19] A. Botchkarev, and P. Andru. *A Return on Investment as a Metric for Evaluating Information Systems: Taxonomy and Application.* Interdisciplinary Journal of Information, Knowledge, and Management 6, pp 245-69, 2011.

A Glossary of Poker Terms

This appendix contains definitions and brief explanations of all used poker terms in this bachelor's thesis. A full glossary can also be found on the web, following the link: https://en.wikipedia.org/wiki/Glossary_of_poker_terms

- ***9-handed***: A game of poker where 9 players are sitting at the table.
- ***Ante***: A forced bet in some variants of poker (especially in tournament poker) that every player has to pay in order to receive cards.
- ***Bet***: A bet describes the amount of chips a player puts into the pot in a betting round when the pot it unopened.
- ***Bet size***: The amount of chips wagered in a betting action.
- ***Betting action***: Describes the type of action in a betting situation. For example: Bet, 2-Bet, 3-Bet, 4-bet.
- ***Betting turn***: needed?
- ***Big blind***: A forced bet that the player two positions to the left of the dealer has to pay before receiving his cards. The Big blind is usually twice the size of the small blind.

- ***Blind structure***: This is a poker tournament specific term, which describes the temporal structure of blinds. Blinds are usually increased in fixed time intervals.
- ***Blinds***: Force bets that are split into small blind and big blind. The player immediately to the left of the dealer has to pay the small blind, the player two positions to the left of the dealer has to pay the big blind in order to receive cards.
- ***Buy-In***: Describes the amount of money payed to enter a tournament and gain chips.
- ***Call***: Matching a bet made by other players in the current betting round.
- ***Cash game***: It is a poker variant where the chips at the same time represent real money value. The blinds are staying the same over time.
- ***Check***: The action of staying in a hand without committing any money to the pot. This is only possible if the betting round is unopened.
- ***Community board***: Describes the shared cards on the table as an entity.
- ***Community card***: Face up dealt cards on the table, shared between all players.
- ***Dealer***: Either a person on the table dealing cards to players or the position on the table holding the dealer button.
- ***Dealer-button***: An object marking the seat on the table that receives the last card dealt pre-flop.
- ***Flop***: The first three cards dealt face up to the community board. It is followed by the second betting round.
- ***Fold***: The action of laying down/ giving up a hand.

- ***Forced bets***: Bets that have to be payed in order to receive cards pre-flop. This includes the small blind, the big blind and antes.
- ***Hand***: The cards dealt to a player.
- ***Hand rank***: The relative strength of a hand at showdown. In Texas Hold'em poker the best 5-card combination defines the hand rank.
- ***Heads-up***: When only two players are competing in a hand.
- ***Hole cards***: The two face down cards dealt to a player pre-flop.
- ***Late position***: Describes the two players sitting on the dealer position and one seat to the right of the dealer (the so-called "Cut-Off" position").
- ***Limit poker***: A poker variant where there are certain minimum and maximum limits on bet sizes.
- ***Long run***: Describes a lengthy time period.
- ***No-Limit poker***: A poker variant where there are no minimum or maximum limits on bet sizes.
- ***Pre-flop***: A betting round before the flop. In the pre-flop stage of a hand player are being dealt cards and have the chance to bet.
- ***Position***: The position on the table relative to the dealer seat. Player close to the left of the dealer are in early position and players close to the right of the dealer are in late position.
- ***Pot***: The accumulated amount of chips wagered by all players combined, that is awarded to the winner of the hand.
- ***Pot odds***: The ratio between the size of the pot and the amount of chips needed to call.
- ***Probability triple***: Describes a probability distribution across the three

possible betting options *bet/raise, check/call, fold*.

- ***Raise***: Raising a bet means first matching a bet made by an other player and increase that bet by a certain amount.
- ***River***: The fifth card dealt face up to the community board. It is followed by the fourth and final betting round.
- ***Showdown***: If more than one player remains after the last betting round (River), the player's hands are exposed and a winner is determined.
- ***Small blind***: A forced bet that the player immediately to the left of the dealer has to pay before receiving his cards.
- ***Stack***: The total amount of chips or money a player has currently available to play on the table.
- ***Suited***: When both hole cards share the same suit
- ***Turn***: The fourth card dealt face up to the community board. It is followed by the third betting round.