FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Reinforcement Learning applied to the game of Poker

José Pedro Marques

Mestrado Integrado em Engenharia Informática e Computação

Co-Supervisor: Henrique Lopes Cardoso Ph.D

Co-Supervisor: Luís Paulo Reis Ph.D

June 18, 2013

Reinforcement Learning applied to the game of Poker

José Pedro Marques

Mestrado Integrado em Engenharia Informática e Computação

June 18, 2013

Abstract

Games have always been a field of interest for Artificial Intelligence, as Games have usually simple rules. However in order to play them at a competent level, some degree of complex strategies are required. Despite this, some good (and well known) advances were made, specially in games like Chess or Checkers.

But while these advances were important, results in games like Chess or Checkers - deterministic games with complete information - are hard to adapt to real world problems, as it is rare to have situations with complete information, and real world problems usually have stochastic variables. Thus the research has shifted into stochastic games with incomplete information, like Poker. There have been some excellent results, and recently a computer agent was able to win against a world class human player, the culmination of years of research and many different approaches.

While these are indeed excellent results, the focus of this dissertation is somewhat different. It is not intended to create a world class Poker player, but one agent that can learn how to play a game from scratch. This approach could lead to more generic architectures, allowing them to be ported to another games more easily. In this work, a Reinforcement Learning approach was pursued, and the Poker variant Texas Hold'em was used as a test bed.

This work shows two different agents that were capable of learning the game from scratch, achieving positive results versus a great variety of adversaries. It shows the learning structure and the decisions made during the development of such agents, concluding that while this is a viable approach to the game of Poker, the more generic the agent is, the worse it performs.

Resumo

Os jogos sempre foram um campo de interesse para a Inteligência Artificial, visto que normalmente têm regras simples e fáceis de entender, mas no entanto para os jogar a um nível competente são necessárias estratégias com algum grau de complexidade. Apesar disto, vários avanços foram feitos em jogos como o Xadrez e as Damas.

Apesar destes avanços serem importantes, resultados em jogos como o Xadrez e as damas – jogos determinísticos com informação completa – são difíceis de adaptar a problemas reais, visto que problemas com informação completa são raros, e no mundo real existem sempre variáveis aleatórias. Sendo assim, o foco da pesquisa alterou-se, e passou se a focar em jogos estocásticos com informação escondida, como o Poker. Nesta área já foram obtidos resultados excelentes, e recentemente um agente foi capaz de ganhar num jogo de Poker contra um jogador de classe mundial, o culminar de muitos anos de pesquisas e de muitas abordagens diferentes.

Enquanto estes resultados são verdadeiramente bons, o foco desta dissertação é algo diferente. Não é o objetivo desta dissertação criar um agente que jogue Poker a uma classe mundial, mas sim criar um agente capaz de jogar um jogo sem conhecimento prévio das regras do mesmo. Esta abordagem poderá levar a arquitectura mais genéricas, que poderão ser adaptadas para outros jogos e domínios. Neste trabalho, uma abordagem baseada em Aprendizagem pro Reforço foi seguida, e a variante de Poker "Texas Hold'em" foi utilizada como jogo base.

Este trabalho mostra dois agentes diferentes, que foram capazes de aprender o jogo do zero, obtendo resultados positivos contra uma variedade de adversários. Mostra a estrutura de aprendizagem e as decisões feitas durente o desenvolvimento, concluindo que, enquanto esta é uma abordagem viavél ao jogo de Poker, quanto mais genérico for o agente, pior é a sua performance.

Acknowledgements

To make this work possible, all external collaboration was essential, which I would like to thank.

First, I would like to thank Prof. Dr. Henrique Lopes Cardoso and Prof. Dr. Luís Paulo Reis for their help and guidance. I owe this dissertation's quality to their patience and constant revisions.

I would also like to thank Msc Luís Teófilo for all the help given, specially regarding the game of Poker.

Finally I would like to thank to my family and friends. They always have supported me on the most difficult times, and nothing would have been possible without their support.

José Pedro Marques

"It is impossible to live without failing at something,
unless you live so cautiously
that you might as well not have lived at all - in which case, you fail by default."

J.K. Rowling

Contents

List of Figures

List of Tables

Abbreviations

AI	Artificial Intelligence
CPRG	Computer Poker Research Group
CFR	Counterfactual Regret Minimization
IRC	Internet Relay Chat
LIACC	Laboratory of Artificial Intelligence and Computer Science of the University of Porto
RL	Reinforcement Learning
RNR	Restricted Nash Response

Chapter 1

Introduction

> "If every tool, when ordered, or even of its own accord, could do the work that befits it then there would be no need either of apprentices for the masters or of slaves for the lords" - Aristotle

The concept of an artificial intelligence is not new, as even Aristotle could foresee use for an autonomous thinking machine that could solve problems, but it was not until 1956, in a conference at Darthmouth College that the field of Artificial Intelligence (AI) research was coined [McC04].

At the beginning, the main objective of AI research was to develop a strong AI i.e. an intelligence that matches or exceeds human intelligence. With the passage of time, it was obvious that this goal was, if not unattainable, at least harder then expected, and the focus of AI research shifted to solving smaller, particular problems. This is commonly called weak AI.

With this shift in focus, several new branches on AI were formed, and many exciting results were found. One area that benefited from this split was Games, as games have inherent properties that make them attractive for this kind of research [Bil06]. In general, games have **well defined rules**, but require **complex strategies**. Some even among the hardest problems known in computational complexity and theoretical computer science. Furthermore, games have a **clear and specific goal**, with a clear definition of success, and allow **measurable results**, either by the degree of success in playing the game, or in solutions to related subtasks.

But not all games are alike, and there are two major discriminants to classify games: the **amount of information available** and the **presence of randomness**. Games where players have access to all the information they require about the game during play are classified as having **perfect information**. On the other hand, if some information is indeed hidden from the player, the game is known as having **imperfect information**. An example of a game with perfect information is Chess, while Minesweeper is an example of a game with imperfect information.

Furthermore, games can be classified as **stochastic** or **deterministic**. If a game contains an element of chance, such as the shuffling of cards, that introduces randomness into the game, this game is known as a **stochastic** game while the absence of this element ensures that the game is **deterministic**. An example of a stochastic game is backgammon, while Chess and Checkers are examples of deterministic games.

There have been great breakthroughs on the field of AI, but mainly focused on *deterministic* games, with *perfect* information, such as chess [CHH02] and checkers [SLLB96]. While these excellent results are encouraging, they lack applicability to real world problems, as in most situations it is required to deal with uncertainty and chance.

1.1 Motivation

The challenges brought by non-deterministic games with hidden information are much closer to those present in real world applications and therefore a solution that can be applied to a game with those characteristics is one that can possibly be adapted to solve real world problems.

This is where Poker enters the scene, as a *stochastic* game with *imperfect information*. It is stochastic because the shuffling of cards introduces randomness into the game and it is a game of imperfect information, as players cannot see their opponents' cards, forcing players to make decisions based on hidden information. Given the relatively simple rules of the game there is an enormous amount of subtle and sophisticated scenarios that can occur during a hand of play (this is particularly true of the Texas Hold'em variation). Moreover, Poker is an inherently psychological game. It is crucial to have an understanding of your opponents and how they think to be able to play well. Therefore Poker ensures that issues such as probabilistic reasoning and opponent modelling need to be considered, in order to build a competent artificial player. All these factors make Poker a challenging domain for AI related research where advances are likely to be beneficial outside the realm of poker itself.

Aside from computer science, Poker has also aroused interest in other fields such as economy [Liv08] and psychology [CEW98].

1.2 Goals

The goal of this project is to create an agent capable of learning how to play a full game of Poker (Texas Hold'em variation) from scratch. This approach should be as generic as possible, in order to apply the same methodology to other (similar) games, and eventually port the methodology to other domains, not necessarily game-related.

One important part of the project (directly related to the game of Poker), is to ascertain the impact of a good abstraction, in order to reduce the amount of game states to a tractable number.

Another key goal is to ascertain how generic an agent can be, and still perform well. This was tested comparing the two developed agents versus a variety of adversaries and comparing their results.

1.3 Dissertation Structure

This document is divided into six chapters. In this first chapter, a small introduction of the problem is made, presenting the problem, the motivation that led to this work and the goals that were pursued.

In the second chapter some key concepts of Computer Poker research are presented, such as **Information Sets** and **Abstraction**. Also a survey of the state of the art on Computer Poker research is presented, showing computer poker techniques and the most promising results of Computer Poker research. Then a small description of the available tools is made, making a brief reference to related projects.

The third chapter describes the game of poker and its rules, focusing more on the Texas Hold'em variant. Poker's importance to the artificial intelligence field is also explained.

The fourth chapter presents the approach taken to solve the problem, exposing the developed architecture, the agents developed and their learning structure.

In the fifth chapter the concept of abstraction is analysed. Four different bucketing strategies are compared, and the results of the comparison experiments are shown.

The sixth chapter presents the results obtained after the experiments of each developed agent. It shows the results of each agent playing against 5 other bots, comparing and explaining the results obtained.

Finally, in the seventh chapter, the main achievements are highlighted and conclusions are drawn, as well as possible future work in this area of research.

Chapter 2

State of the Art

Research on Computer Poker has been active for well over 20 years, with most of the contributions coming from the University of Alberta Computer Poker Research Group. Several different approaches were made during the years, with each approach improving upon the previous ones mistakes. Though there have been many breakthroughs, only recently were computer agents able to win against a professional Poker player. In the following sections a brief overview of the most common techniques used in Computer Poker will be given, followed by a description of a few of the most notable approaches in how to build a poker playing computer agent. Finally a brief introduction to reinforcement learning is given, and the most common tools used in computer poker are shown.

2.1 Computer Poker Techniques

Poker is a very complex game, and every basic strategy needs to take into account the stochastic nature of the game and the understanding of the opponents.

Artificial Poker players can easily compute probabilities and expectations rather easily, but Poker itself is a very complex game. Mathematically, the Texas Hold'em variant has $3 * 10^{17}$ game states. In order to determine the best play in any given situation, it is necessary to reduce this search space, by creating *abstractions*, that reduce the game to a simpler version, trading a loss in accuracy for a gain in feasibility.

2.1.1 Information Sets

Information sets are a consequence of the hidden information present in games like Poker, as there exists a set of game states that we cannot distinguish between them, due to the hidden information, as can be seen in figure 2.1. This set of game states are called *information sets* and, as it is obvious, they are far less than actual game states (in the Texas Hold'em variant there are 10^{18} game states and $3.19 * 10^{14}$ information sets)

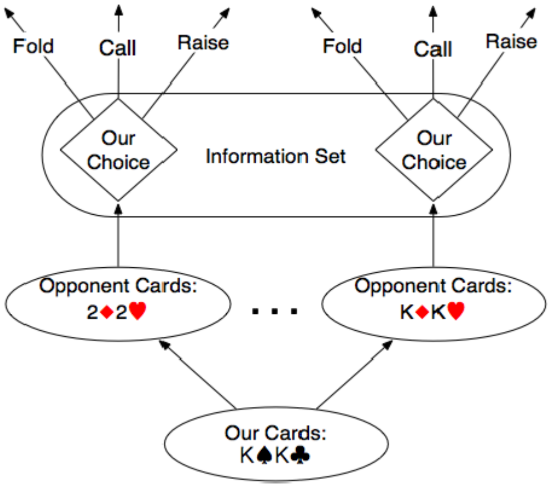

Figure 2.1: An example of an information set

As we cannot distinguish game states inside the same information set, it follows that for every information set we will play in the same way - there is a single strategy. This strategy is known as a *behavioural strategy*: a probability distribution over actions for each information set.

2.1.2 Bucketing

The idea behind bucketing is to split cards with similar value into "buckets", so we can easily reduce the number of situations we need to check. This is quite similar to the way humans play the game. For example, the strategy doesn't really change much if a player is holding a King and a 4 or a King and a 3. These are pairs of cards that could go into the same bucket - "King and low card". It is clear then that one difficulty with this method is the hand evaluation. How can I score the hand I have? And secondly, one single value may not be enough to fully characterize one hand.[Bil06]

It is clear then that the quality of the abstraction depends directly on the number of buckets we use. The greater the number of buckets, the closer the abstraction is to the real game, and therefore strategies calculated in a better abstraction are bound to perform better, as they can recognize a greater number of different game states.

In this kind of abstraction, a strategy is defined over a sequence of buckets, instead of a sequence of cards. For example, if before the flop my cards were in the bucket 1 (weak cards,e.g. 2 and 6) and with the 3 flop cards they are much stronger (e.g. 2 6 2 on the flop), they change to bucket 5. So the strategy will take into account the sequence 1-5 after the flop. To find the probability for every transition various techniques can be used. With the probability to move from each bucket at a betting round to another bucket on the next betting round, the model is built and the pseudo-optimal strategy created.

Nevertheless it is a powerful method of abstraction and advanced bucketing techniques have been developed and used successfully in several different artificial poker players.[Bil06] [Joh07]

2.1.3 Opponent Modelling

Any competent Poker player must be able to react to the games condition and adjust to the opponent thus opponent modeling is often considered a cornerstone of any competitive poker agent, and any agent that implemented an opponent modelling module has had an improvement in its results (see section 2.2) and has been the focus of extensive research [Hoe06] [CF08]

Limited observations, imperfect information, and dynamic behaviour are the main challenges presented to opponent modelling and although these challenges are not yet completely resolved, there are techniques to handle them. Currently two popular approaches (machine learning models) are used to build an opponent's model in poker: decision-trees and artificial neural networks. Both are based on statistic observation and both require a substantial amount of good data in order to be accurate.

2.2 Building a Poker Playing agent

Currently there are three main types of approaches to build a poker playing agent: *heuristics* based, *simulation* based and *game-theoretic* based. These approaches focus on creating a strategy or policy for an agent to play, or give an indication of how the agent should react when facing some situation. A brief description of these approaches is given in the next subsections.

2.2.1 Early Poker Research

The first investigation into Computer Poker is credit to Nicolas Findler [Bil95], with some attempts to apply machine learning principles to the game of 5-card draw Poker. He created both static (do not adapt to the opponent) and learning (adapted to the opponent) players, but Findler's analysis of these agents' quality of play has been questioned [Bil95] due to its lack of scientific rigour.

Also based on the game of 5-card draw, Waterman investigated the use of production rules to make a betting decision given information such as the amount of money currently in the pot, the belief measure that an opponent could be susceptible to a bluff, the number of cards the opponent replaced (used in deducing which possible hand the opponent may hold) and a measure of how conservative the opponent is believed to be. Waterman reported that the use of a small set of production rules "produces play at roughly the same level of skill as an experienced human player" [Wat70]

2.2.2 Heuristic Based Systems

A heuristic/rule-based system approach to computer Poker uses various pieces of information to inform a betting strategy. For example, information such as a player's hole cards, the current community cards, the player's current position at the table and the previous betting history of the hand may form part of some heuristic which dictates whether the player should fold, check/call or bet/raise when it is his turn to act.

2.2.2.1 Rule Based

This approach tries to emulate the way human players describe how they play, defining a series of conditional rules and specifying which action should be taken for each situation that arises. While this might sound reasonable at first, given the complexity of a game like Poker, this approach tends to over-simplify situations, in order to make them computationally feasible, and has been proven extremely limited. [Bil06]

2.2.2.2 Formula Based

Building upon the rule based approach, a formula based approach resorts to a complex procedure (or formula) to try and distinguish situations. This can greatly enlarge the number of identifiable situations, thus being a more flexible approach than the rule based systems. Despite being more flexible, it still suffers (to a lesser extent) from the same liabilities that the rule based approach suffers, as they are both based on the same principle. Furthermore, a system built based on this approach is difficult to maintain and expand, as more abstractions are added to the system.

A good example of an agent using this approach is Loki. Loki was the first poker agent implementation made the University of Alberta CPRG. Loki used a probabilistic formula-based approach, incorporating the GNU poker library [43] high-speed hand comparators as a core function, and expert systems designed by the author, to play (differently) each stage of the game. [BSS99]. To test Loki's performance, the agent participated in an on-line poker game running on the Internet Relay Chat (IRC) [BPSS98].

Though its results were not bad, Loki had several limitations. It required extensive expert knowledge, the lack of adaptive play meant that once a flaw in the strategy was found, it could be exploited without consequence, and it suffers from the complexity of this approach, being difficult to expand and has complicated multi-component systems.[BSS99]

2.2.3 Simulation Based

A simulation-based strategy is analogous to selective search in perfect information games such as chess and checkers, but rather than expanding all nodes in the game-tree with equal probability, biases towards expanding certain nodes is introduced in the hope of obtaining better information in less time by initially examining important nodes. Unfortunately, in practice, this approach can be highly volatile and result in extremely unbalanced play, since the quality of the simulations depends on the quality of the simulated play. This makes it vulnerable to too much biased values for certain situations, leading to inaccurate plays [Sch06].

Poki is an improved version of the previously mentioned *Loki*, based on simulation techniques coupled with opponent modelling. At the beginning of a simulation trial opponents are assigned two hole cards to inform their possible future actions. Instead of assigning cards to different opponents with uniform probability, a weight table is maintained for each opponent which lists all possible two card holdings and the likelihood that those cards would have been played to the current stage in the game. After observing a betting action from an opponent the weights are updated using a probability triple generated by the current opponent model. The use of the weight table allows the assignment of hole cards to an opponent to be biased towards certain cards, rather than assuming equal probability. For example, if an opponent has consistently been raising in a hand it is more likely that the opponent has a good hand rather than a random one.

While *Poki* had better results than *Loki*, results showed that this is still a weak approach, as it would lose against a competent adversary who consistently switched strategies. [BDSS02]

2.2.4 Game Theoretic Based

Game Theory is a branch of mathematics and economics that is devoted to the analysis of games.One very important concept derived from Game Theory is the concept of **Nash Equilibrium**.

Nash Equilibrium is a strategy in which a player cannot do any better by changing his or her strategy provided that their opponents are also using an optimal strategy, moreover a player can reveal their optimal strategy yet not be vulnerable to exploitation by their opponent. This concept can be illustrated fairly simply with the popular game of *rock-paper-scissors* [BDSS02].

The game of *rock-paper-scissors* (or RhoShamBo) is a two player game where each player chooses either rock, paper or scissors. This choice is then made known to both player simultaneously, and the winner is found using the simple rules depicted in figure 2.2. If an equal decision is made, the result is a draw.

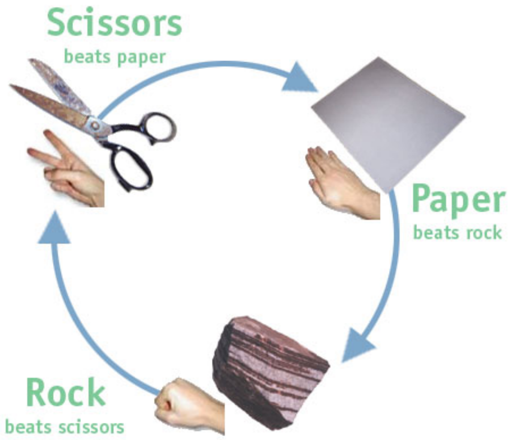

Figure 2.2: Rules to determine the winner of a Rock-Paper-Scissors game

The skill in the game of rock-paper-scissors comes from a player knowing their opponent so well that they are able to predict which option their opponent will choose, i.e. they know their opponents' strategy. This simple game has an obvious *Nash Equilibrium*, or *optimal strategy*, which says to pick randomly with equal probability between rock, paper and scissors. If a weak player deviates from this strategy, a strong opponent can outplay this player by finding out and *exploiting* their *sub-optimal* strategy. However, if the weak player never deviates from this strategy he can never be exploited by the strong player, as he cannot predict what action the weak player will take. Therefore, by following the optimal strategy, the weak player ensures an even result, instead of a losing one. Note that even if the strong player was told about this strategy, it would not change anything.

Unfortunately, creating a perfect Nash equilibrium strategy for a complex game like Poker is extremely difficult and, at the moment, computationally infeasible. Instead what is currently used is ε -Nash Equilibrium, an approximation to the Nash equilibrium strategy, resulting in a sub-optimal strategy that proposes a best response instead of the perfect response. ε is the value of the best response to the determined suboptimal strategy and is a measure of how far from the actual equilibrium the strategy is. If an opponent, either human or machine, makes mistakes then the equilibrium strategy can win over time.

The amount of agents created with this strategy at their core is the best testament to its effectiveness. There is the *PsOpti* family, a series of artificial players designed to play heads-up limit Texas Hold'em.

The PsOpti family of strategies is created by converting the abstract extensive game into a sequence form. The sequence form can then be transformed as a series of constraints in a linear program and be solved to find the approximation to the Nash equilibrium. However, the linear programming required to solve the entire abstract game was considerable and additional simplifications like abstractions and/or separating the game into two phases were used. The techniques used to build this type of agents are described in detail in [BB03]

One good result to note is the fact that PsOpti4 and PsOpti6 were combined to form Hyperborean06, the winner of the 2006 AAAI Computer Poker Competition [LZ06]. Also noteworthy is the fact that PsOpti4 was licensed to commercial product Poker Academy under the name of SparBot.

Even non-institutions have created agents following these concepts, as is the case of *Bluffbot*, a creation of Teppo Salonen. Bluffbot is a combination of an expert system and a game theoretic pesudo-optimal startegy [Sal]. It's main core is a hand-made game theoretic equilibrium strategy based on the expertise of the author. The latest version, BluffBot 3.0 placed second in the no-limit Texas Hold'em series of the third AAAI Computer Poker Competition in 2008.

2.2.4.1 Counterfactual Regret Minimization

One characteristic of the ε-Nash Equilibrium strategies is that they are only as good as the abstraction they are computed in, meaning that the larger the abstraction (i.e. the closer to the real game) the better is the strategy found. The drawback is that the approaches of PsOpti and Bluffbot require memory linear to the number of game states, thus a larger abstraction requires more memory.

To tackle this problem, Counterfactual Regret Minimization (CFR) was proposed [Joh07]. This approach also finds an ε-Nash Equilibrium strategy but requires memory linear to the number of *information sets* rather than the number of game states. As we have previously seen, Poker has

$3.16 * 10^{17}$ game states and $3.19 * 10^{14}$ information sets, thus CFR is able to solve much larger abstractions than were possible with the previous methods [Joh07].

The core idea behind this approach is to play T games of Poker, updating the strategy at each round, and at the end find the best possible strategy that could have been used for all those T games. With that strategy (the best strategy) we can compute the *average overall regret*, as the difference between the value of the best strategy and the value we used. Now if we minimize the *average overall regret*, the *average strategy* used over those T games will approach a Nash Equilibrium strategy.

A computer bot built with this approach was able to beat every previously bot made, mainly because it used abstractions bigger than those that could have been computed before [Joh07]. The drawback of this approach is that it can be seen as a "playing not to lose" strategy, as any Nash Equilibrium strategies. While these are a good starting point, and valuable on their own, in Poker it is necessary to exploit the opponent's flaws, and win by the bigger margin as possible, thus there is a need to explore other strategies that are capable of stepping away from the equilibrium point in order to try to exploit an opponent.

2.2.4.2 Best Response

A best response to a strategy S is the strategy that maximally exploits S. Knowing this strategy and the utility of using it against S confers several advantages. For example, if S is an ε-Nash equilibrium strategy, the utility tells us how far S is from a Nash equilibrium. If we are trying to use another strategy to defeat S, comparing that strategy's utility to the utility of the best response tells us how much of the possible exploitation we are achieving.

However, calculating the best response in usually computationally infeasible which means that we are forced to compromise and calculate an **abstract game best response**. This strategies give us a lower bound on the exploitability of S; we know that the best response can achieve at least the same utility against S as the abstract game best response. [Joh07]

Although promising, the abstract game best response approach has requirements that limit its use in poker games like Texas Hold'em. Firstly, the algorithm requires knowledge of how the strategy acts at each stage of the game; so unless the opponents play in a predictable way or chooses to provide details regarding their own strategy it is difficult to calculate an abstract game best response to an opponents' strategy. Secondly, as was discussed previously, the abstract game best response has to be calculated in the same abstraction as the original strategy, reducing its utility. Finally, it is still a static strategy, and while it counters one strategy, if the opponent switches strategies this approach can not adapt to it.

2.2.4.3 Restricted Nash Response

As we have previously seen, exploiting opponents is important (we'd like to win more money than the Counterfactual Regret Minimization strategies do) but Best Response strategies, while winning by a larger margin than CFR, are brittle, and lossy badly against the wrong opponent. As a compromise between these two approaches Restricted Nash Response (RNR) was proposed.

The main idea behind this approach is that the opponent may have 2 strategies: one weak (static) strategy that they use with *p* probability and a strong strategy that they use with *1-p* probability. With this logic in mind, our agent has two goals: exploit the weak strategy and defend against the strong one. This probability *p* can be seen in two ways: how much you care about exploiting the static strategy or how confident you are that the opponent will actually use the static strategy. Thus if *p* is low, our counter strategy will approach a Nash Equilibrium strategy, while if *p* is high we will approach a best response strategy.

Results obtained with an agent developed with this strategy show its robustness[Joh07]. Comparing to the Best Response approach, it didn't win by as much in the games where Best Response won, but they still win against those that BR lost by much. When compared to a CFR in the same abstraction, it was noted that the RNR counter-strategies, used against the correct opponent, perform better than CFR. Although CFR5 often outplays the RNR counter-strategies against the other opponents, it does not do so by a large margin.

The drawbacks of this approach are still important though.Firstly, Restricted Nash response counter-strategies require a large quantity of observations. It is intuitive that, as any technique is given more observations of an opponent, the counter-strategies produced will grow in strength. Secondly Restricted Nash response counter-strategies are sensitive to the choice of training opponent, as they need observations of all of the game tree. [JB09]

One very noteworthy mention is the bot Polaris. Polaris is a set of agents that the CPRG developed to enter the Man vs. Machine contest promoted by the University of Alberta in 2007. The first version of Polaris had agents that used ε-Nash Equilibrium startegies, Restricted Nash Response and an experimental aggressive ε-Nash equilibrium strategy [Joh07]. While it lost the contest against two professional Poker players, post game analysis and player comments suggested that the day where computer bots would be able to win wasn't that far away [Joh07].

In 2008 Polaris II was featured in the second edition of the Man vs. Machine contest. In this year, Polaris played against six professional players and Polaris became known as the first artificial poker player to beat a team of professional human players. This version of Polaris was subject to much improvement from the previous version and featured new technique, based on importance sampling, which greatly improved the accuracy of its estimators [BJ08].

2.2.4.4 Adaptive Programs

As it is expected, a computer agent playing with a single static strategy will eventually lose, as it cannot react to a change in the opponent's style of play. Therefore to be able to play poker at a world class level, a player must be able to adjust to any special circumstances.

In determining the correct mathematical play to make, adaptive programs can be seen as simply computing a best response but on current opponent's beliefs, which are subject to change over time. In principle this is correct but these advanced systems also refrain from continuously using the best response available, deviating from a simple best response for added benefits, like avoiding predictability, thus avoiding exploitation.

While this approach has several advantages when compared with the previously discussed approaches, it has some problems, like the amount of data required to create a useful model of the opponent is normally high, and a good starting point, as an unlucky stream of initial games might lead to (extremely) wrong beliefs. [Dav02]

Based on this approach two bots were built: Vexbot and its successor BRPlayer. The main advantage of Vexbot and BRPlayer is their ability to learn online and adapt their play to exploit their opponent. In experiments performed by Billings and colleagues [BDS04], Vexbot was shown to defeat PsOpti4, and to defeat a variety of opponents by a much greater margin than PsOpti4 did. This shows the value of an adaptive strategy over a Nash equilibrium strategy. As was mentioned previously, exploitation is important: the goal is not necessarily to not lose , but to win as much as possible from each opponent.

One very important aspect of both these bots, related directly with the focus of this work, is that at the beginning of the game, neither knows the rules of poker, and they learn them by observation. [Joh07]

2.2.4.5 Data Biased Response

This approach was developed to address the issues found in the Restrict Nash Response approach. The main difference in this approach is that instead of using a single parameter p for choosing which strategy to use for every information set like in RNR, in Data Biased response exists a one probability for **each** information set. What this means in practice is that when a player reaches an information set I, it tries to exploit the adversary with P(I) probability and tries to defend (use a Nash Equilibrium strategy) with 1-P(I). This results in a better model, as we can change P for every information - P(I)- according to the number of observations we have for that information set, thus creating a more accurate model. This approach was found to solve many of the problems presented by the RNR approach [JB09]. Firstly, data biased response doesn't require a default strategy. If no observation were made, any other strategy can be played although ideally it should revert to a self-play ε-Nash equilibrium strategy. Secondly, it embodies "quality-assurance" since it sets a minimum number of observations in order to express any confidence in the model's accuracy while implementing linear and curve confidence functions for a trustworthy assessment of the model's accuracy.

2.3 Reinforcement Learning

Reinforcement learning is learning what to do–how to map situations to actions-so as to maximize a numerical reward signal. The learner is not told which actions to take, as in most forms of supervised learning, but instead must discover which actions yield the most reward by trying them. In the most interesting and challenging cases, actions may affect not only the immediate reward but also the next situation and, through that, all subsequent rewards. These two characteristics–trial-and-error search and delayed reward–are the two most important distinguishing features of reinforcement learning.[SB98]

In machine learning, the environment is typically formulated as a Markov decision process (MDP), and many reinforcement learning algorithms for this context are highly related to dynamic programming techniques. The main difference to these classical techniques is that reinforcement learning algorithms do not need the knowledge of the MDP and they target large MDPs where exact methods become infeasible.

The basic reinforcement learning model consists of:

- a set of environment states ;
- a set of actions ;
- rules of transitioning between states;
- rules that determine the scalar immediate reward of a transition;
- rules that describe what the agent observes;

A reinforcement learning agent interacts with its environment in discrete time steps. At each time T, the agent receives an observation O_t, which typically includes the reward R_t. It then chooses an action a_t from the set of actions available, which is subsequently sent to the environment. The environment moves to a new state S_{t+1} and the reward r_{t+1} associated with the transition (S_t, a_t, S_{t+1}) is determined. The goal of a reinforcement learning agent is to collect as much reward as possible. The agent can choose any action as a function of the history and it can even randomize its action selection.

2.3.1 Reinforcement Learning applied to Poker

There have been a few attempts to develop an agent using a Reinforcement Learning methodology to poker. One approach for an agent that plays only in the pre-flop stage of the game was designed [Pas11].This approach consists of a Q-Table containing the state-action pairs. Each state σ is define as:

- **G**: A value representing the pair of cards the player holds;
- **P**: The player's position on the table;
- **T**: The opponent type;
- **A**: A value representing the last action taken;

To each state there exists a corresponding tuple (C,R) where C is the probability of calling and R the probability of raising ($C+R \leq 1$). Therefore, when the agent plays, it searches the Q-Table to obtain the values of C and R and a random number ($N \in [0,1]$) is drawn. The action is chosen as follows:

$$Action = \begin{cases} Call & : N \in [0,C] \\ Raise & : N \in]C, C+R] \\ Fold & : N > C+R \end{cases}$$

Results of applying the approach show promise. While this agent was designed to play only in the pre-flop stage, it beat every adversary [Pas11], thus showing that this is a valid starting point to create a complete Texas Hold'em agent.

2.4 Tools

As the research on computer poker grows, a number of tools were develop to help and sustain this growth. In this section we will describe those that are most useful for the work in hand.

2.4.1 Open Meerkat testbed

Poker Academy was, probably, the world's most popular commercial software trainer for Poker. Based on the research by the University of Alberta CPRG, it is a Texas Hold'em program that simulates the experience of playing in a real online poker room. It simulates both limit Hold'em and no-limit Hold'em, ring and tournament play. Its added value comes from the world renown AI used for its bots specifically Sparbot, Vexbot (for heads-up limit games) and Poki (for ring limit games). The most noteworthy feature this software offers though is the Meerkat API, a Java API for developing bots that can be placed in the Poker Academy environment and tested against the bots developed by the CPRG. Unfortunately Poker academy is no longer active, but the API remains.

Open Meerkat testbed is an open source implementation of the Meerkat API for running poker games. It imitates the late Poker academy simulator, but without an User INterface, thus making it much faster to run AI vs AI games. It generates bankroll evolution plots; implements seat permutation (replay games with same cards but with different seat order) and generate game logs. It also shows online bankroll evolution graph

2.4.2 LIACC's Texas Hold'em Simulator

The Laboratory of Artificial Intelligence and Computer Science of the University of Porto (LIACC) was created in 1988 to promote the collaboration of researchers that were separately working in the fields of Computer Science and Artificial Intelligence in different Faculties. At present the LIACC researchers belong to the Computer Science Department of the Faculty of Science, to the Informatics Engineering Department of the Faculty of Engineering, to the Informatics Engineering Department of the University of Beira Interior, and to the School of Engineering of the Polytechnic Institute of Porto. [LIA]

This center has developed a Poker Simulation system called "LIACC's Texas Hold'em Simulator". This system has a client/server architecture and provides two default clients : One that can be controlled by humans and one autonomous agent that runs with an incorporated AI poker playing algorithm, based on the betting strategy described in [Bil06].

The implemented Server is capable of supporting ten different Clients (using a standard poker protocol based on TCP/IP) and allows the user to define all major characteristics of a poker game (initial stack, the value of the blinds) and its protocol is compatible with the Meerkat API.

Chapter 3

Poker

Poker is a popular type of card game where players bet that their hand is stronger than the hands of their adversaries. All bets go into the pot and when the game ends, the player with the best hand wins the current pot.

3.1 Betting

Knowing when, and what to bet is the key in most poker games, as it allows to minimize losses and maximize gains. Every time a bet is made, that money is deemed lost to everyone, except the final winner. When it reaches the player's turn, he can choose his action from the following list:

- **Check**: A check can be seen as a bet of 0, and therefore can only be made if no one has made a bet in the current round.
- **Call**: To call is to place a bet of equal value of the current maximum bet.
- **Bet**: A bet is a wager of any amount of money (or chips). This can be limited by the rules of the game. Once a player bets, the remaining players need to atleast match the bet to continue in the game.
- **Raise**: To Raise is to place a bet higher than the current maximum bet. This enforces the remaining players to at least Call the bet if they want to continue in the game. Again, the maximum amount that can be raised may be limited by the game rules.
- **Fold**: Folding means leaving this round without placing any further bets. Any player who folds loses the right to claim the final pot.
- **All-In**: A special case of raise, where the player wants to call but doesn't have sufficient money (or chips) to match the current maximum bet. He instead bets all his money and can only win what was bet until that moment.The rest of the pot, when the game ends, belongs to the second place. After a player goes All-In, no further bets are placed, and all betting rounds are skipped until the end of the game.

3.2 Texas Hold’em

At the moment, the Texas Hold’em variant of Poker is the most popular game of its kind, and therefore is the variant that will be considered in this work. After its introduction in 1920, the Texas Hold’em game had a slow rise in popularity until 1998, were it had a boost with the release of the film "Rounders". The film tells the story of a poker player and its adventures in the underground poker world. Due to the dramatic representation of the swing that a poker player faces in this game, it became very appealing to people all around the world. Over the next two years, due to the sudden interest in the game, a lot of literacy had come to the stores, available to anyone.

3.2.1 Rules

In Texas Hold’em players are dealt two cards each (hole cards), and a maximum of 5 community cards are placed on the table. Any combination of 5 cards from the two hole cards and the 5 community card is called a hand. Besides this, the position of a player in the table is important to the flow of the game. There is always a dealer, and depending on the number of players, a big blind and a small blind. These positions are usually marked with a chip and are rotated clockwise after each round.

Figure 3.1 shows a normal table, with player F being the dealer and players A and B the small and big blind respectively.

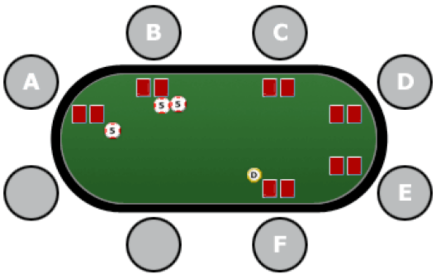

Figure 3.1: Texas Hold’em positions

Before any card is dealt, both the small and big blinds bet a predetermined amount of money to the pot. The big blind is the minimum bet for the table, and the small blind is half of that. This ensures that there is a starting pot, so there is always an incentive to play.

The game then is comprised of 4 rounds, and each round ends when all players go all-in, call or check. The four rounds are:

- **Pre-Flop**: no community cards;
- **Flop**: three community cards are dealt;
- **Turn**: the forth community card is dealt;

- **River**: the fifth and final community card is dealt.

After the river, if all players agree to call/check the pot, it's time for the **showdown**. In the showdown the players start showing their hand sequentially, and all the players except the first have a choice of not showing his hand (and therefore losing the pot). The player with the best hand wins the pot. If two or more players have similar ranked hands, there is a tie and the pot is divided.

In each round if all players fold except one, the remaining player is the winner and can collect the pot.

3.2.2 Hand Ranking

There are some general rules that apply to each hand, regardless of rank.

- Individual cards are ranked A, K, Q, J, 10, 9, 8, 7, 6, 5, 4, 3, 2.
- Suits have no value. The suits of the cards are mainly used in determining whether a hand fits a certain combination (specifically the flush or straight flush hands).
- A hand always consists of five cards. In games where more than five cards are available to each player, the best five card combination is the hand of those cards players
- Hands are ranked first by combination, then by individual card rank: even the lowest qualifying hand in a certain combination defeats all hands in all lower combinations. The smallest two pair hand, for example, defeats all hands with just one pair or high card. Only between two hands in the same category are card ranks used to break ties.

The following card hands combinations represent the poker hand ranking, in descending order of strength:

Royal Flush is the best possible hand in standard five-card poker. Ace, King, Queen, Jack and 10, all of the same suit.

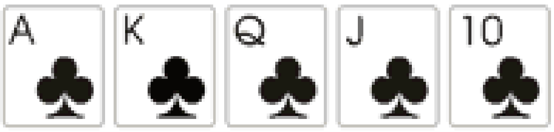

Figure 3.2: Example of a Royal Flush

Straight Flush The Straight Flush is a hand containing five cards in sequence, all of the same suit. When two players go to showdown with this hand, the hands are compared by the highest ranked card. Aces can play high or low in Straights and Straight Flushes - A-2-3-4-5 is a straight with 5 as a high card and 10-J-Q-K-A is a straight with Ace as a high card.

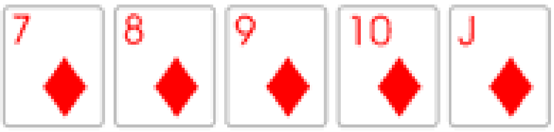

Figure 3.3: Example of a Straight Flush

Four of a Kind is a hand where the player controls four cards of a certain rank and an unmatched card of another rank. As always, if two players have the same four of a kind, the one with the highest fifth card -the kicker- wins.

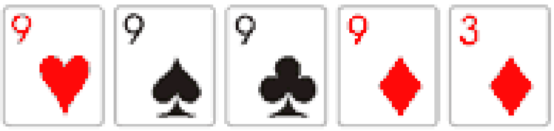

Figure 3.4: Example of a Four of a Kind

Full House is a hand where all the cards are active. It contains a set of three matching cards of the same rank, and two matching cards of another rank. Between two full House hands, the one with the highest ranking set of three wins.

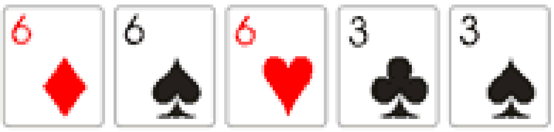

Figure 3.5: Example of a Full House

Flush is a hand containing five non sequential cards of the same suit. If two players have this hand, the one with the highest card wins. If both have the same higher card, then it compares the second higher, and so on until a difference is found. Since the suits are not used to rank the cards, if both players have the same cards, with different suits, they are tied.

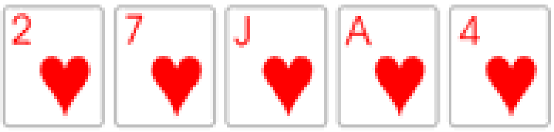

Figure 3.6: Example of a Flush

Straight is a hand containing 5 sequential card ranks, but with different suits. In case of two Straight hands, the one with the higher ranked card wins. If the higher ranked cards of both hands are of the same rank, the hands are tied since the suit doesn't differentiate them.

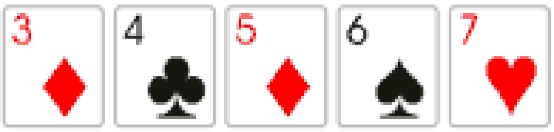

Figure 3.7: Example of a Straight

Three of a Kind is a hand containing three cards of the same rank and two unmatched cards. A Three of a Kind hand wins another if the set has a higher ranked card. In case of both sets containing the same ranks, the higher ranked of the unmatched cards are compared to break the tie.

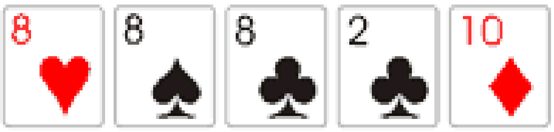

Figure 3.8: Example of a Three of a Kind

Two Pair is a hand containing two cards of the same rank, another pair of matched rank, different from the first, plus an unmatched card. To rank two hands of this type, the higher ranked pair of each is compared, and the higher one wins. In case both players have the same higher ranked pair, the second pair is compared. Finally, if both hands have the same two pairs, the kickers determine the winner. There is also a possibility of a tie if both hands have the same ranked cards.

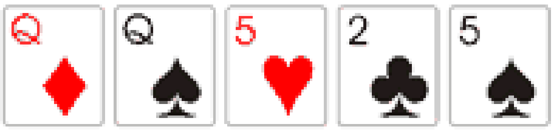

Figure 3.9: Example of a Two Pair

One Pair is a hand containing two cards of the same ranks and three other unmatched cards. In a Showdown with two hands containing One Pair, the one with the higher ranked pair wins. If the pair is of the same rank in both hands, the kickers are compared in descending order until it is possible to determine the winner. If both pairs and kickers are of the same rank in both hands, the game is tied.

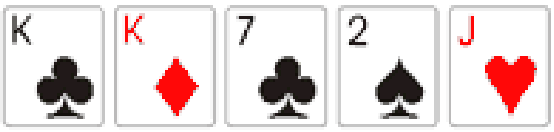

Figure 3.10: Example of a Pair

High Card is a combination in which no two cards have the same rank, the five cards are not in sequence and all cards are not of the same suit. Two High Card hands are ranked comparing the highest ranking card. If those are equal then the second highest ranking card is compared, and so on until a difference is found. If both hands are similar in rank, the game is tied.

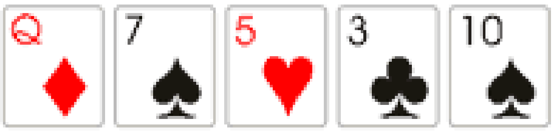

Figure 3.11: Example of a High Card

3.3 Importance to artificial intelligence

Strategic games can be classified by two parameters: deterministic and information completeness.

A game being deterministic means that the next state of the game will be purely defined by the current player, without external factors. If a game is not deterministic, it means that there are external factors that influence player decision, such as random events. A game where random events influence the game flow can be also called stochastic.

Regarding information completeness, a game with complete information is a game where, at any stage of the game, anyone can identify the whole state of the game (i.e. there are no hidden variables). Therefore, a game with incomplete information is one where only partial information is given to each player, making it hard (or impossible) to predict the consequences of any given action.

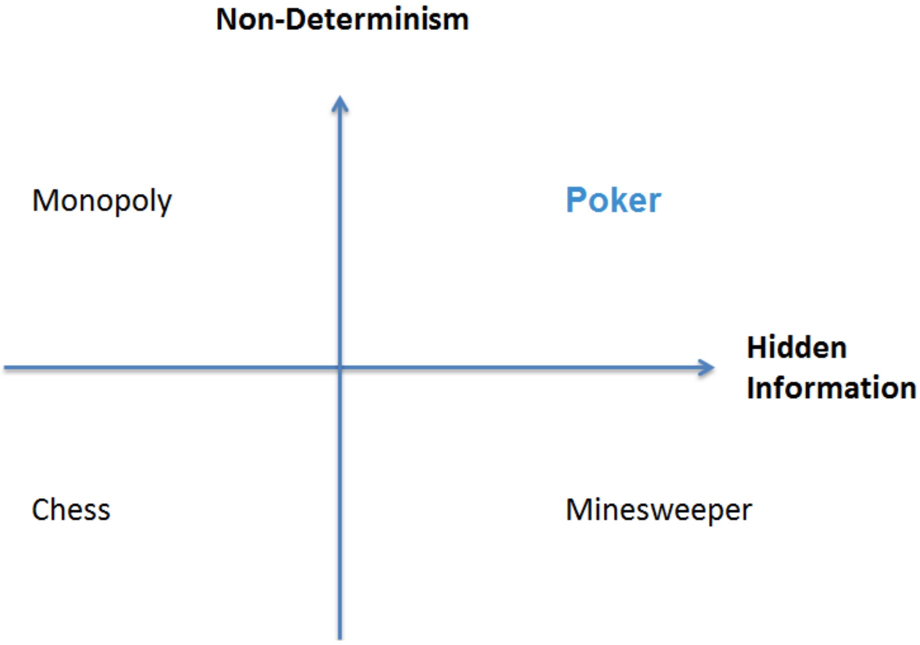

Figure 3.12: Game classification

As can be seen in figure 3.12, Poker classifies as a stochastic game (the randomness is introduced each time the deck is shuffled) with partial information (my opponent cards are hidden to me).

These characteristics make Poker an excellent challenge to artificial intelligence, when compared to other games such as chess. To play Poker a player must make predictions of other players, with incomplete information about them. These mechanisms to deal with misinformation are a complex challenge and with proper adjustment they can be used in other branches of science.

Chapter 4

Poker agents overview

In this chapter it is presented the developed agents specification, including the general agent's architecture and all the necessary development steps to implement it. Two agents were developed, with the same general architecture but with different learning methodologies.

4.1 Developed Agents

During this work, two agents were developed, to test two different learning methodologies. The first, **Outcome Learner** tries to learn when to play based on the known outcomes. The second, **Action Learner** tries to learn the best action for each situation. The differences between these two agents will be made explicit in the following sections.

4.1.1 Outcome Learner

The main idea behind an Outcome Learner agent development was to create the most generic agent as possible, while still playing poker. In order to do so, this agent has a Q-Table containing the state-action pairs. Each state σ is defined by:

- **Bucket - B**: An integer representing the current bucket our cards are in.
- **Last Stage - LS**: An integer representing the last played stage. This allows the agent to distinguish between first betting round of a stage and subsequent betting rounds on the same stage.
- **Last player type - LPT**: An integer representing the type of the last player who acted on this stage.
- **Last player action - LPA**: An integer representing the action of said player. This last two integers incorporate the opponent modelling into the state, thus the agent can learn to play against a myriad of opponents.
- **Position - P**: An enumeration describing the position of the player. This can be either Early, Mid, or Late. This allows the agents to adjust their play style based on their position, as a

Poker player playing last has a clear advantage as he has already seen the other players play and has more information. So actions should be different depending on where our agent is sitting, and this value helps distinguish those situations.

Each state has a direct correspondence to a pair (**W** - wins, **L** - losses):

$$\sigma(B, LS, LPT, LPA, P) \to (W, F) : W, F \in [0 \cdots +\infty); \tag{4.1}$$

$$B \in \{0 \ldots \ldots N\}; \tag{4.2}$$

$$LS \in \{-1, 0, 1, 2, 3\}; \tag{4.3}$$

$$LPT \in \{0, 1, 2, 3\}; \tag{4.4}$$

$$LPA \in \{0, 1\}; \tag{4.5}$$

$$P \in \{'Early', 'Mid', Late'\} \tag{4.6}$$

These definitions allow our agent to reduce the 10^{18} information sets that a full Poker game has to a potential $N \cdot 5 \cdot 4 \cdot 2 \cdot 3 = 120N$ information sets, with N being the amount of buckets that our bucketing strategy employs.

The Q-Table is initially empty and the weights are initialized as 0. Therefore, in order to gain some initial knowledge, a random strategy was defined. While acting on this random strategy, the agent will call with a 70% probability and raise with 30%. The rationale behind the choice to not include a fold action in the random actions is simple: folding leads to the least information gain available. If our agent folds, and the winner does not show his cards, we find ourselves in a situation where we cannot learn anything. Thus in the beginning, when the agents needs to learn, we will either call or raise, ensuring we arrive at a final state that we can learn from.

After the amount of observation surpasses a pre-defined minimum, the agent will start to build its strategy. At this phase, when the agent plays, it searches the Q-Table to obtain the values of **W** and **L** so as to decide on the action to take. After retrieving these values, the following values are calculated:

$$winProbability = \frac{W}{W + L} \tag{4.7}$$

$$foldProbability = min(1, e^{-3.5(winProbability - 0.05)}) \tag{4.8}$$

$$playProbability = 1 - foldProbability \tag{4.9}$$

$$callProbability = \frac{1}{4} \cdot playProbability \tag{4.10}$$

$$raiseProbability = \frac{3}{4} \cdot playProbability \tag{4.11}$$

In Figure 4.1 we can see the plotting of the folding function $e^{-3.5(winProbability - 0.05)}$. On the horizontal axis we have the wining probability and on the vertical axis we have the folding probability. Note that for a winning probability of 1 we still have a folding probability of 5%. This ensures that our agent's strategy isn't too rigid and adds some exploration space.

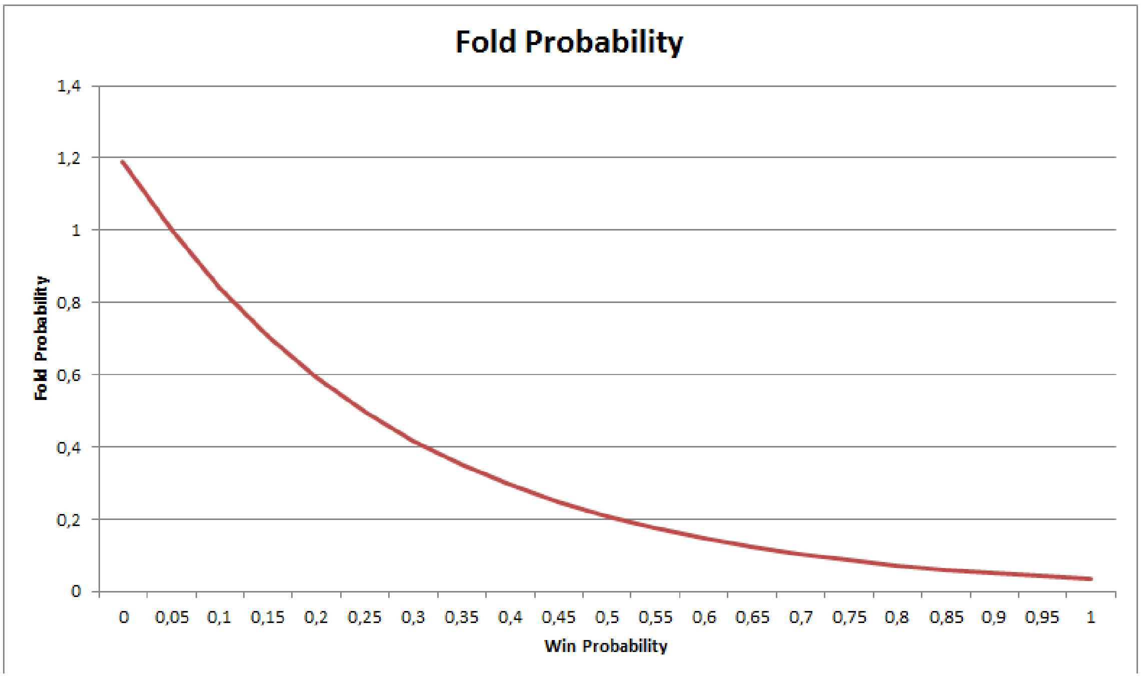

Figure 4.1: Plot of the folding probability function

When we reach the end of each game, both Q values (W and L) for every visited state are updated based on the outcome, regardless of the action taken. This decision is based on the fact that an action taken on an early stage is as important as an action taken in the last stage, and contributed as much to the end result, so every state is updated.

$$W_{t+1} = W_t + 0.9 * (R + max(W_{s+1}) - W_t) \tag{4.12}$$

$$L_{t+1} = L_t + 0.9 * (1 - R + max(L_{s+1}) - L_t) \tag{4.13}$$

Where R is 1 for a win and 0 for a loss. $max(L_{s+1})$ and $max(W_{s+1})$ represent the maximum for these weights on information sets for the next stage of the game.

A learning rate of 0.9 was chosen due to the volatile nature of a Poker game. If an opponent switches strategy, it is required that the agent can adapt to this change, and learn a new counter strategy.

4.1.2 Action Learner

An Action Learner is a less general implementation of the previous agent. Instead of deciding whether or not to play, this agent tries to learn the correct action to take in each state.

In order to do so, while maintaining the same state definition, the (**W** , **L**) pair was replaced by the triple (**F** - fold weight, **C** - call weight, **R** - raise weight). Thus:

$$\sigma(B, LS, LPT, LPA, P) \rightarrow (F, C, R) : F, C, R \in [0 \cdots +\infty); \tag{4.14}$$

$$B \in \{0 \ldots\ldots N\}; \tag{4.15}$$

$$LS \in \{-1, 0, 1, 2, 3\}; \tag{4.16}$$

$$LPT \in \{0, 1, 2, 3\}; \tag{4.17}$$

$$LPA \in \{0, 1\}; \tag{4.18}$$

$$P \in \{'Early', 'Mid', Late'\} \tag{4.19}$$

The weights on the Q-Table are initially filled up with random numbers, except for the fold weight, which is initialized at 0, following the same rationale as before.

When we reach the end of each game, all Q values (F, C and R) for every visited state are updated based on the game outcome, the action taken and the amount of money won or lost. The rationale behind this decision is simple: if we lost, maybe we should have folded sooner. Conversely, if we won, maybe we should have raised sooner. Thus every action that led to the game's outcome is analysed and each state's Q values are updated.

$$Q_{t+1} = Q_t + R(Outcome, Action, AmmountWon) \tag{4.20}$$

Where Q_t represents the Q values for the given state and $R(Outcome, Action, AmmountWon)$ is a function that calculates the reward based on the outcome of the game, the action taken and the amount of money won or lost. This function is shown in table 4.1, where each arrow corresponds to the amount won (or lost) in the game *(*X). Therefore $\uparrow\uparrow$ means that this value will be incremented by *2X* and $\downarrow$ means that the value will be decremented by *X*.

As table 4.1 there was a clear focus on regret minimization, as the (negative) reward for not folding is usually larger than the rest.

When deciding on which action to take, the agent looks up the Q values (**F**,**C**,**R**) for the current state and they are normalized to the $[0 \ldots 1]$ range as follows:

$$f = \frac{F}{F + C + R} \tag{4.21}$$

$$c = \frac{C}{F + C + R} \tag{4.22}$$

$$r = \frac{R}{F + C + R} \tag{4.23}$$

Action Taken	Game Outcome	Action Chosen	Reward
Fold	Win	Fold	↓
		Call	↑
		Raise	↑↑
	Loss	Fold	↑↑
		Call	↓
		Raise	↓
Call	Win	Fold	↓
		Call	↑
		Raise	↑↑
	Loss	Fold	↑↑
		Call	↓
		Raise	↓↓
Raise	Win	Fold	↓
		Call	↓
		Raise	↑↑
	Loss	Fold	↑↑
		Call	↑
		Raise	↓

Table 4.1: Rewards given to the states based on the outcome and action taken

A random number $rnd \in [0\ldots1]$ is then generated and an action is chosen based on the following rules:

$$Action = \begin{cases} Call, & rnd \in [0\ldots c] \\ Raise, & rnd \in]c\ldots c+r] \\ Fold \quad otherwise \end{cases} \tag{4.24}$$

4.2 Bucketing module

In each round we split the cards we currently control (our two hole cards plus the available community cards) into a fixed number of **buckets**, keeping hands with similar strategic properties in the same bucket. One approach for doing this is to divide hands into buckets based on their strength, such that weak hands are grouped into low numbered buckets, and strong hands are grouped into high numbered buckets.

The **bucket sequence** is therefore the sequence of buckets that the player's cards were placed into in each round. So if a player has a strong hand in the Preflop that turns into an even stronger hand in the Flop we can see a bucket sequence of "5-10", meaning that he had cards in bucket 5 in the Preflop and in bucket 10 in the Flop. Strategies are then defined by the bucket sequence, as a hand that progresses from bucket 5 to bucket 10 is notoriously different from one that progressed from bucket 5 to bucket 1.

This is one of the essential modules for both agents, as it allows to greatly reduce the number of poker states into a manageable few, for instance if we select a small number of buckets, we reduce the 1326 possible combinations of Preflop cards to a manageable number. As a consequence, strategies defined using this abstraction may no longer be capable of optimal play in a real game.

As the number of buckets will be defining the size of our information sets, and thus are directly related to the number of distinct situations our agents can recognize (the same strategy will be employed to all the hands in the same bucket) it is important to make the most of the buckets we choose to use. Therefore, we would like to sort our hands into buckets such that all of the hands in each bucket are strategically similar.

As mentioned earlier, one straightforward way to partition hands into buckets is to consider the "strength" of the hands. To calculate this **Hand Strength** we use a metric called **7 card hand rank** [Joh07]

The main idea behind this technique is to count how many times we end up ahead (or tied) given the cards we have now, and those that could be dealt.

To do so, we start by completing the remaining board cards, drawing random cards from the deck. After we have the full board, we then iterate through all possible 2 hand cards combinations that one opponent might hold and we count the percentage of times we win (or tie). We then repeat this experiment for a fixed number of random boards, in order to reduce the variance of the **hand strength**.

For example, if we are at a Flop stage, we have our own two cards, plus 3 already on the board. To calculate our hand strength we repeat the following steps a fixed number of times N:

1. Draw 2 random cards from the deck. This completes the board cards.

2. For all possible 2 card combinations, and the complete board, count the number of times we win (a tie counts as half a win).

3. $hand \quad strength_i = \frac{wins}{\binom{52}{2}}$

Thus our expected hand strength ($E[HS]$) is

$$E[HS] = \frac{\sum_{i=1}^{N} hand \quad strength_i}{N} \tag{4.25}$$

Parallel to this metric, we also find the hand's expected strength squared ($E[HS^2]$) [Joh07]. This metric assigns higher values to hands that have more potential. For example, with the hand strength method previously describe, if one hand (hand 1) reaches two hand strength value of 0.5 its strength ($HS_1 = \dfrac{0.5+0.5}{2} = 0.5$) is the same hand strength as one (hand 2) that reaches one value of 0.8 and one of 0.2 ($HS_2 = d\frac{0.8+0.2}{2} = 0.5$). We could argue that these hands are different. Hand 2 seems to have more potential to improve, thus we'd like to treat them differently. With the hand strength squared method, Hand 1 would have a $HS_1^2 = \frac{0.5^2+0.5^2}{2} = 0.25$ while hand 2 would have a $HS_2^2 = \frac{0.2^2+0.8^2}{2} = 0.68$ and now these hands would be treated differently.

Then our hand strength squared ($E[HS^2]$) is

$$E[HS^2] = \frac{\sum_{i=1}^{N} (hand\quad strength_i)^2}{N} \tag{4.26}$$

Thus we have two different variables $E[HS]$ and $E[HS^2]$, both in the $[0\dots1]$ range to classify the cards in our hand. We now need a function to map these values to buckets. In this work, two different approaches were implemented: Linear Bucketing and Nested Bucketing, combined with two different functions - Uniform and Non-Uniform - to create four different strategies.

4.2.1 Uniform versus Non-Uniform partitioning

Every bucketing strategy needs a function $f(b,N)$ that given a bucket b and a total number of buckets N returns a number in the $[0\dots1]$ range: the maximum value that goes into b. Having that, $E[HS]$ or $E[HS^2]$ values in $[0, f(0,N)]$ fall into bucket 0, $(f(0,N), f(1,N)]$ into bucket 1 and so on. In this work two functions were implemented. The first one splits the $[0\dots1]$ range in a uniform way, i.e. all partitions have the same amplitude. For example, if we use 8 buckets and the $E[HS]$ metric, then each partition has a range of $\frac{1}{8}$ so a hand with an $E[HS]$ of 0.48 will fall in the 4th bucket.

While this is a good way of partitioning the range, the most important hands are those with a higher $E[HS]$ (or $E[HS^2]$), therefore we would like to have diminishing amplitudes on the partitions, allowing us to distinguish between these more important hands. To achieve this goal, non-uniform partitioning was implemented. The function that models this behaviour is

$$f(i,n) = a + (b-a)\sqrt{\frac{i}{n-1}} \tag{4.27}$$

With $a, b \in [0\dots1]$ and $i, n \in \mathbb{N}$. a and b define respectively the maximum and minimum values for the curve. In Figure 4.2 we can see a plot of this function with $a = 0.3$, $b = 1$ and $n = 12$ and in Figure 4.3 $a = 0.2$, $b = 0.8$ and $n = 10$. As we can see, as the number of the bucket increases (x-axis) we start getting smaller amplitudes on the partitions.

4.2.2 Linear versus Nested Bucketing

Linear bucketing is the most intuitive way of partitioning the hands into buckets. We choose one of the two metrics ($E[HS]$ and $E[HS^2]$) and one of the two functions previously described and we have our bucketing strategy. While this has some valour onto itself, we are limited to a one dimension characterization of hands, which is rather crude.

To solve this problem, Nested Bucketing was introduced [Joh07]. We first split the hands according to $E[HS^2]$ into N buckets and then according to $E[HS]$ into M buckets. This allows us to separate the high potential from the high value hands, producing $N \times M$ buckets.

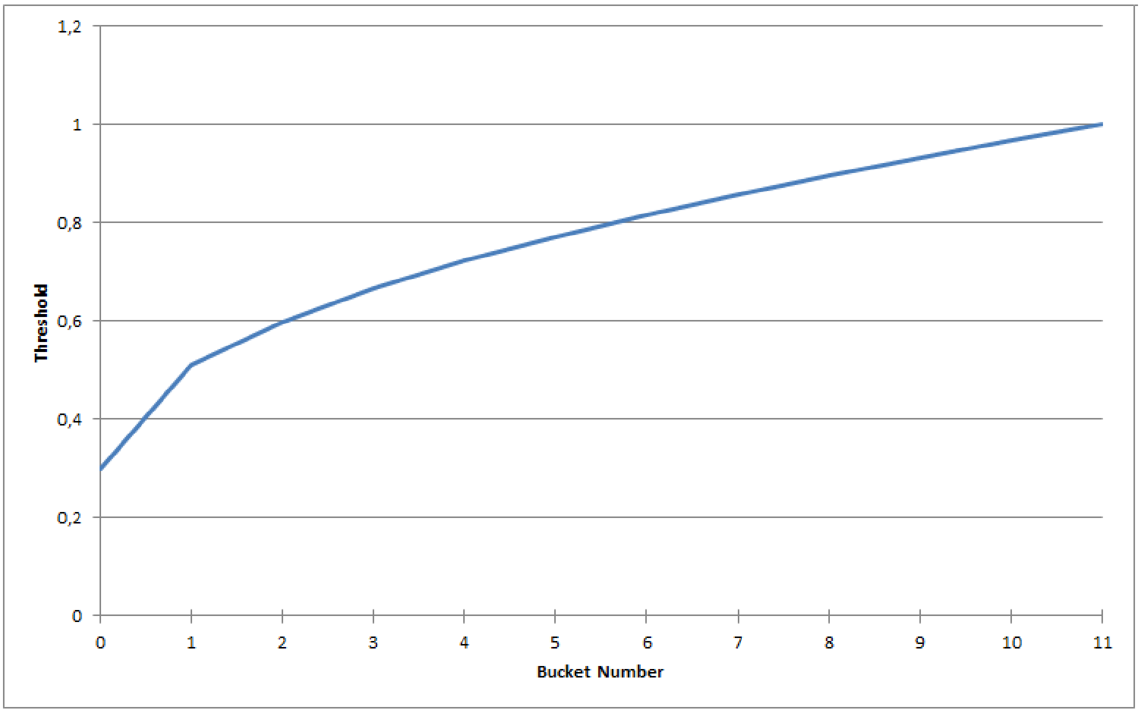

Figure 4.2: Plot of the bucketing non-uniform function with parameters a = 0.3 , b = 1 and n = 12

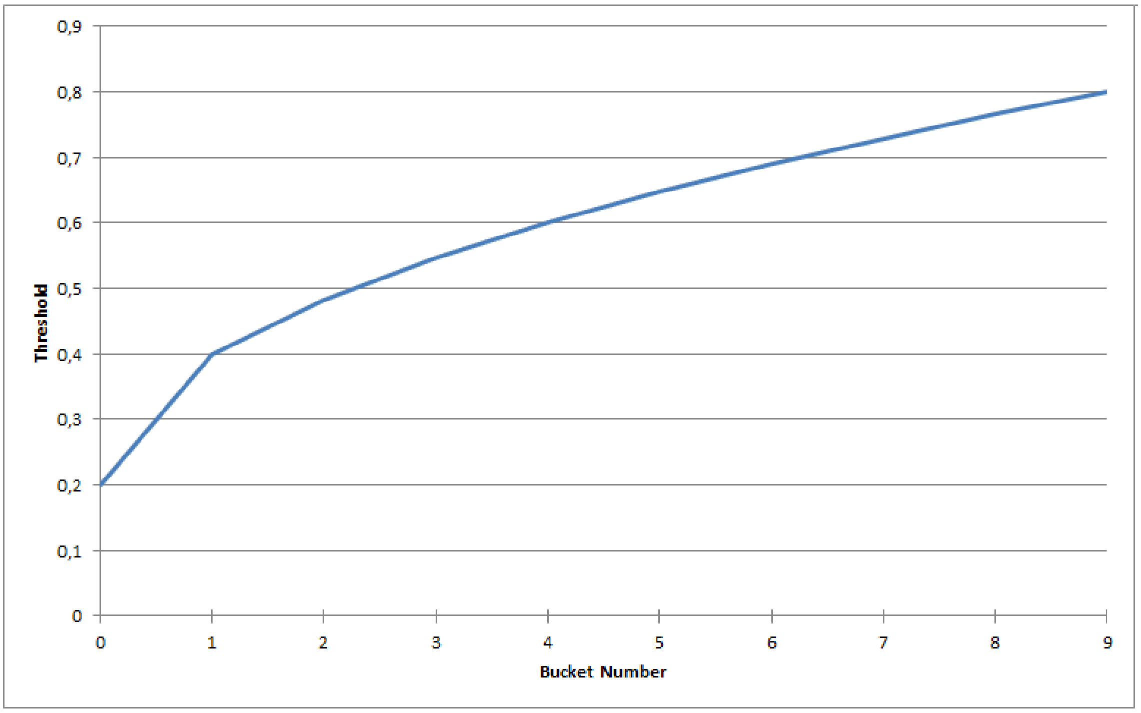

Figure 4.3: Plot of the bucketing non-uniform function with parameters a = 0.2 , b = 0.8 and n = 10

4.3 Opponent Modelling module

The agents possess a simple opponent modelling module, based on player classification. It consists in grouping players with similar play styles. The criteria used for player classification is based on how much and when he bluffs, what kind of hands he bets, how aggressive he is, etc. One of the most known player classification is the Slansky classification [Skl02]. One important characteristic used for player classification is the percentage of hands he plays. A player can be classified as tight if it plays 28% or less hands, and loose if he plays more than 28% of the times [F́08]. Another important characteristic is the aggression factor (AF). Aggression factor is given by the following formula:

$$AF = \frac{nBets + nRaises}{nCalls} \tag{4.28}$$

This formula allows classifying players as aggressive, if AF is greater than 1, or passive, if AF is bellow 1. Aggression factor combined with percentage of played hands leads to four playing styles:

- Loose Aggressive
- Loose Passive
- Tight Passive
- Tight Aggressive

Therefore our agents keep track of each opponent actions and try to categorize them with these rules, in order to learn the best counter strategy versus each opponent type.

4.4 Basic Flow

Figure 4.4 shows the basic flow of the game. The learning process occurs only when the game ends, and resembles the way a human would learn.

Before a game starts the agent tries to load previous knowledge, gained by playing other games, to the knowledge base. If there are not any, the agent starts from scratch (i.e. all states are new).

When the game starts, we are dealt our two hole cards and we query the Bucketing module to see into which bucket those cards fall into, based on the current Bucketing Strategy. That information, coupled with the environment variables consists of a State. We then query the Knowledge base, to see if we have been in this state before. If not, a new state is added to the knowledge base. Based on this state's knowledge we can then proceed to get the next Action we will take. This process is repeated until either we reach the last phase (showdown) or we fold, or everyone else folds.

When the game ends, the agent checks his last action. If he folded (meaning he lost the game) he tries to see if the opponent has shown his cards. If he can, he compares his own cards to his, to see what the outcome would have been if he had not folded. The agent then proceeds to learn, as was specified before.

After this phase, we proceed with the game. If there is another game to be played, we go back to the start and just repeat this process. Otherwise, we write our recently acquired knowledge onto a file, so we can use it later on.

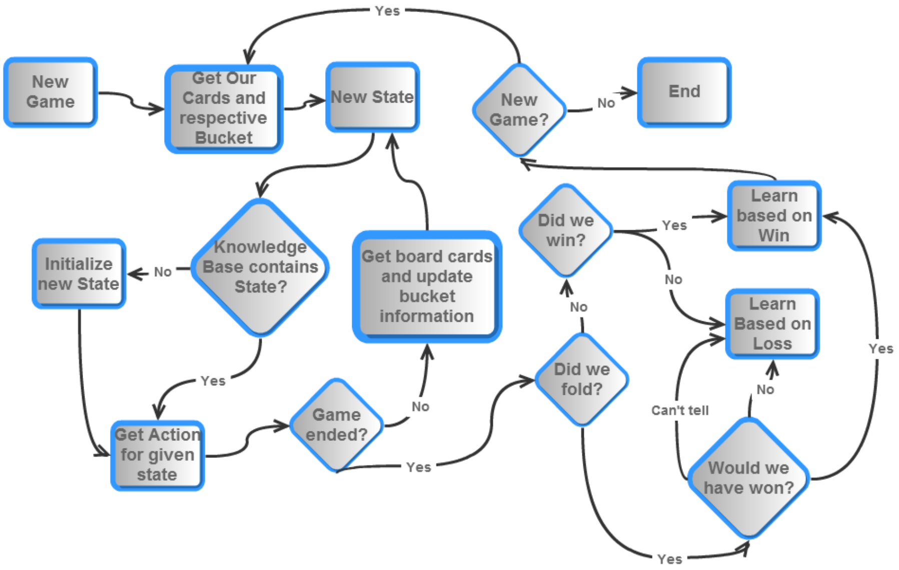

Figure 4.4: General flow of a Poker Game

Chapter 5

Abstraction strategy refinement

In this chapter we explore different strategies to assign hands to buckets, as this assignment has a big impact on every agent game play, seeing that we will employ the same strategy for every hand in each bucket. Thus hands with similar strategic properties need to be assigned to the same bucket.

Another key point to keep in mind is that the number of information sets is directly proportional to the number of buckets we use, which means that increasing the number of buckets , while allowing our agents to potentially differentiate more cases, can have diminishing returns. If the number of information sets is too big, each information set will be seen less times, thus increasing the number of required games for each information set's weights to converge.

The aim of this chapter is thus to provide some insight regarding the most appropriate abstraction policy, both in terms of the number of buckets to use and of the mapping strategy between hands and buckets.

5.1 Buckets usage

The first experiences made were related only to the bucketing module, and the various bucketing strategies developed. In order to do so, all possible hand combinations for the Preflop and the Flop were tested and their corresponding bucket, according to each strategy was noted. Six different strategies were tested, both with 12 (6x2 for the nested) and 24 (8x3 for the nested) buckets. These strategies are detailed in Figure 5.1

5.1.1 Preflop

On the Prelop, there are a total of $\binom{52}{2} = 1.326$ unique hands. The results of splitting them linearly into 12 and 24 buckets are shown in tables 5.1 and 5.2, respectively, both using Hand Strength and Hand Strength squared as metric. For the non-uniform function, the minimum was set at 0.35 and the maximum at 0.88 for the $E[HS]$ metric and 0.25 and 0.8 for the $E[HS^2]$.

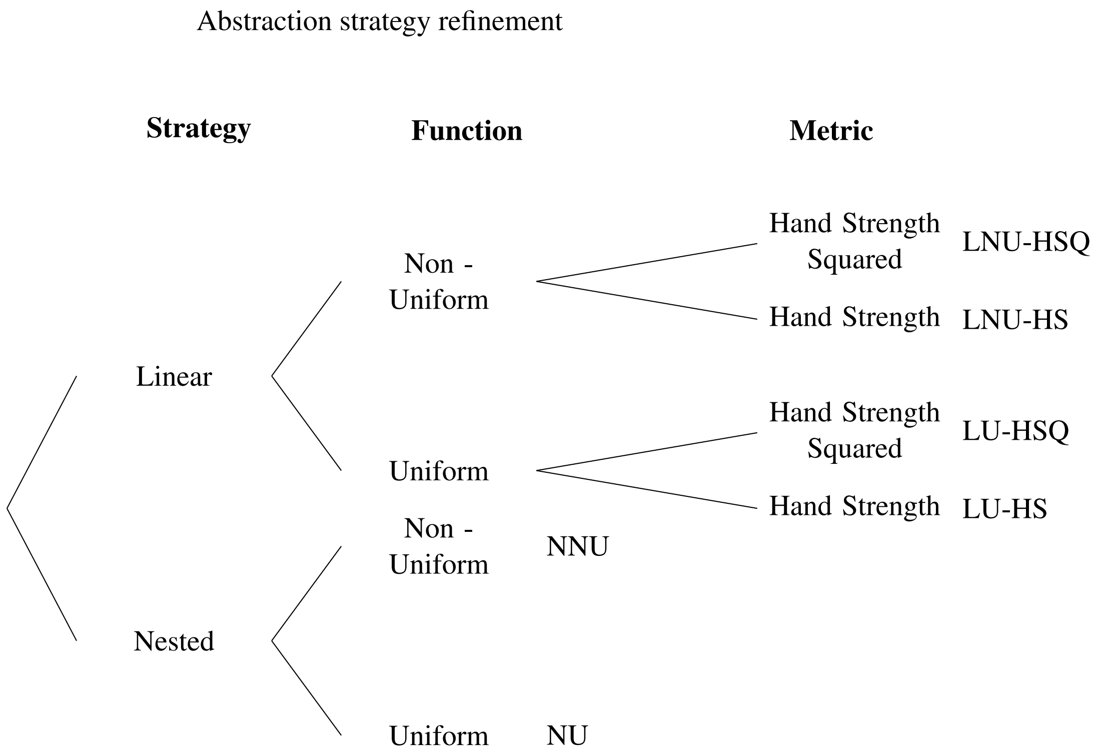

Figure 5.1: Division of the six bucketing strategies, according to splitting strategy, function and metric

Bucket	LNU-HS		LU-HS		LNU-HSQ		LU-HSQ	
	Threshold	**# Hands**	**Threshold**	**# Hands**	**Threshold**	**# Hands**	**Threshold**	**# Hands**
1	0,35	11	0,0833333	0	0,25	590	0,0833333	0
2	0,509801	74	0,1666667	0	0,4158312	270	0,1666667	59
3	0,5759928	171	0,25	0	0,4845208	213	0,25	72
4	0,6267835	229	0,3333333	3	0,5372281	125	0,3333333	430
5	0,669602	224	0,4166667	125	0,5816625	70	0,4166667	445
6	0,7073259	245	0,5	374	0,6208099	20	0,5	256
7	0,7414309	188	0,5833333	418	0,6562019	12	0,5833333	81
8	0,7727937	116	0,6666667	296	0,6887482	9	0,6666667	23
9	0,8019855	26	0,75	72	0,7190416	8	0,75	13
10	0,829403	16	0,8333333	23	0,7474937	8	0,8333333	6
11	0,8553352	11	0,9166667	15	0,7744044	1	0,9166667	0
12	0,88	15	1	0	0,8	0	1	0

Table 5.1: Distribution of the preflop hands into 12 buckets, with 4 different strategies

Figures 5.2 and 5.3 depict the bucket usage for each nested bucketing configuration, where the bubble radius is directly related to the amount of hand per bucket. 4

5.1.2 Flop

On the Flop, there are a total of $\binom{52}{2} = 2.598.960$ unique hands. The results of splitting them linearly into 12 and 24 buckets are shown in tables 5.3 and 5.4, respectively, both using Hand Strength 2

Bucket	LNU-HS		LU-HS		LNU-HSQ		LU-HSQ	
	Threshold	# Hands	Threshold	# Hands	Threshold	# Hands	Threshold	# Hands
1	0,35	11	0,041667		0,25	592	0,0416667	
2	0,4605126	28	0,083333		0,343831	133	0,0833333	
3	0,5062885	44	0,125		0,382698	118	0,125	
4	0,5414135	69	0,166667		0,412521	108	0,1666667	
5	0,5710253	95	0,208333		0,437663	87	0,2083333	7
6	0,5971138	113	0,25		0,459814	82	0,25	77
7	0,6206996	90	0,291667		0,479839	61	0,2916667	181
8	0,642389	98	0,333333	2	0,498255	48	0,3333333	236
9	0,6625769	114	0,375	40	0,515396	31	0,375	228
10	0,6815379	122	0,416667	103	0,531494	19	0,4166667	208
11	0,6994717	98	0,458333	177	0,546721	9	0,4583333	160
12	0,716529	116	0,5	181	0,561204	3	0,5	110
13	0,732827	94	0,541667	201	0,575042	3	0,5416667	57
14	0,748459	79	0,583333	201	0,588314	6	0,5833333	24
15	0,7635004	61	0,625	177	0,601085	7	0,625	6
16	0,7780136	32	0,666667	135	0,613408	3	0,6666667	10
17	0,7920506	18	0,708333	55	0,625326	5	0,7083333	9
18	0,8056553	6	0,75	18	0,636877	2	0,75	8
19	0,8188654	3	0,791667	10	0,648093	4	0,7916667	3
20	0,8317134	8	0,833333	14	0,659002	3	0,8333333	2
21	0,8442275	10	0,875	9	0,669627	2	0,875	
22	0,8564325	6	0,916667	3	0,67999		0,9166667	
23	0,8683502	6	0,958333		0,690109		0,9583333	
24	0,88	5	1		0,7		1	

Table 5.2: Distribution of the preflop hands into 24 buckets, with 4 different strategies

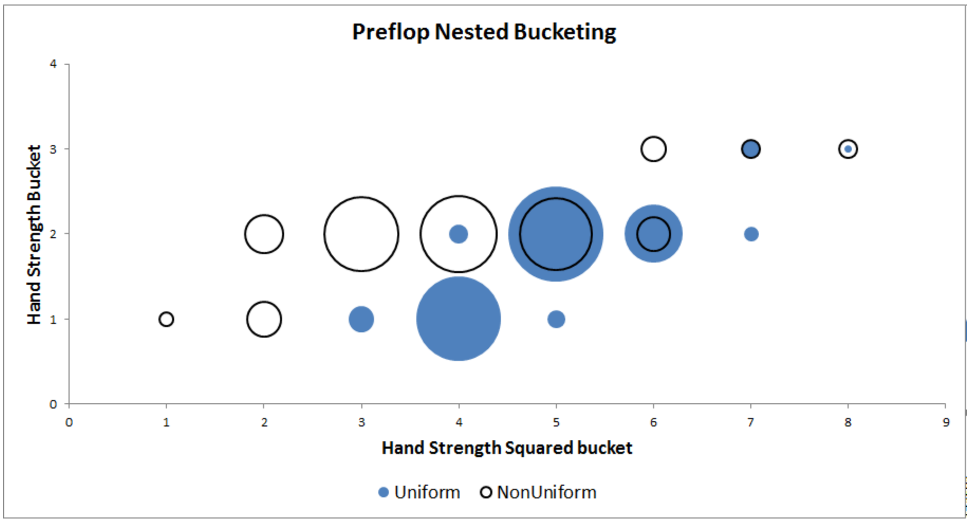

Figure 5.2: Uniform vs Non Uniform split with 6x2 nested bucketing configuration

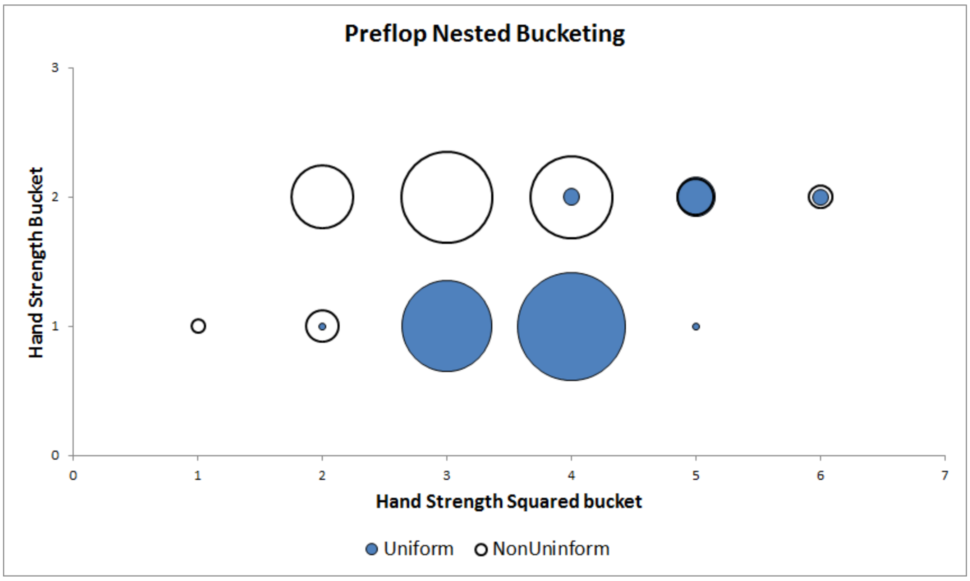

Figure 5.3: Uniform vs Non Uniform split with 8x3 nested bucketing configuration

and Hand Strength squared as metric. Note that for the non-uniform function, the minimum was set at 0.25 and the maximum 1. 4

Figures 5.4 and 5.5 depict the bucket usage for each nested bucketing configuration, where the bubble radius is directly related to the amount of hand per bucket. 2

Bucket	LNU-HS		LU-HS		LNU-HSQ		LU-HSQ	
	Threshold	# Hands	Threshold	# Hands	Threshold	# Hands	Threshold	# Hands
1	0,25	254513	0,0833333	1	0,25	1145348	0,0833333	49901
2	0,4761335	1076500	0,1666667	13907	0,4761335	934679	0,1666667	547910
3	0,5698011	366680	0,25	240605	0,5698011	205581	0,25	547537
4	0,6416747	293216	0,3333333	399431	0,6416747	107335	0,3333333	377092
5	0,702267	193235	0,4166667	408125	0,702267	71502	0,4166667	363660
6	0,7556499	132113	0,5	367112	0,7556499	41161	0,5	255514
7	0,8039117	100220	0,5833333	327456	0,8039117	30160	0,5833333	166199
8	0,848293	72684	0,6666667	318807	0,848293	33393	0,6666667	118538
9	0,8896021	45550	0,75	228396	0,8896021	19758	0,75	75462
10	0,9284005	46277	0,8333333	164727	0,9284005	6063	0,8333333	56248
11	0,9650969	14495	0,9166667	100161	0,9650969	2968	0,9166667	35948
12	1	3477	1	30232	1	1012	1	4951

Table 5.3: Distribution of the flop hands into 12 buckets, with 4 different strategies

Bucket	LNU-HS		LU-HS		LNU-HSQ		LU-HSQ	
	Threshold	# Hands	Threshold	# Hands	Threshold	# Hands	Threshold	# Hands
1	0,25	254187	0,041667		0,25	1144982	0,041667	220
2	0,4063858	758128	0,083333	1	0,406386	699475	0,083333	49811
3	0,4711629	296908	0,125	319	0,471163	222073	0,125	228511
4	0,5208682	197201	0,166667	13567	0,520868	123174	0,166667	318667
5	0,5627716	161341	0,208333	80037	0,562772	83517	0,208333	296230
6	0,5996893	159351	0,25	160263	0,599689	59331	0,25	251543
7	0,6330654	133046	0,291667	195640	0,633065	48449	0,291667	203820
8	0,663758	105993	0,333333	203771	0,663758	41697	0,333333	173725
9	0,6923259	88937	0,375	205196	0,692326	32346	0,375	184804
10	0,7191574	74661	0,416667	202890	0,719157	23808	0,416667	178642
11	0,7445354	61852	0,458333	190398	0,744535	19287	0,458333	143107
12	0,7686731	51871	0,5	177114	0,768673	15643	0,5	112499
13	0,7917363	48155	0,541667	153623	0,791736	13973	0,541667	91819
14	0,8138571	44330	0,583333	173705	0,813857	15211	0,583333	74050
15	0,8351421	35397	0,625	173473	0,835142	16345	0,625	62043
16	0,8556796	27163	0,666667	145673	0,85568	14611	0,666667	56892
17	0,8755432	21440	0,708333	125197	0,875543	10133	0,708333	43505
18	0,8947952	20844	0,75	102931	0,894795	6216	0,75	32017
19	0,9134888	23941	0,791667	87706	0,913489	3406	0,791667	25804
20	0,9316699	18991	0,833333	77136	0,93167	1443	0,833333	30229
21	0,9493786	9142	0,875	50769	0,949379	1117	0,875	25894
22	0,9666498	2757	0,916667	49326	0,96665	1897	0,916667	10153
23	0,9835145	2655	0,958333	26070	0,983514	729	0,958333	3242
24	1	669	1	4155	1	97	1	1733

Table 5.4: Distribution of the flop hands into 24 buckets, with 4 different strategies

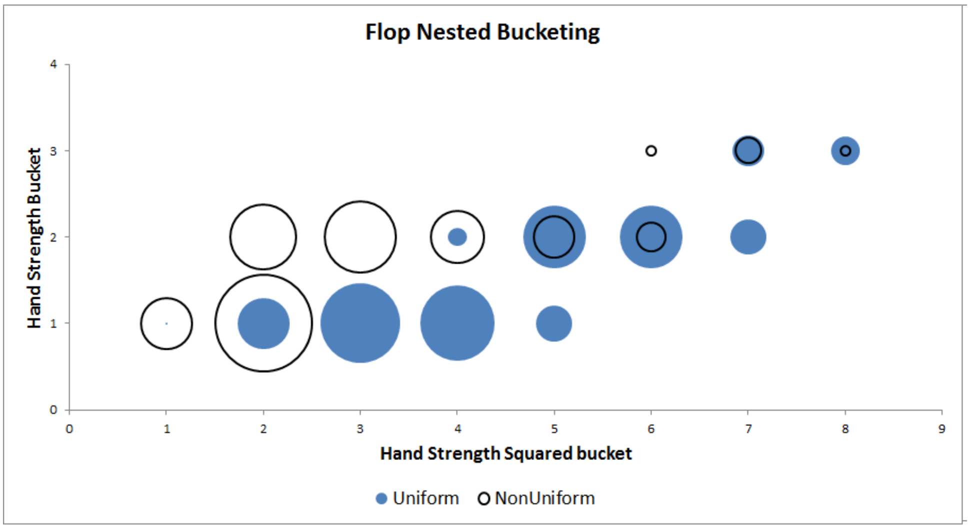

Figure 5.4: Uniform vs Non Uniform split with 6x2 nested bucketing configuration

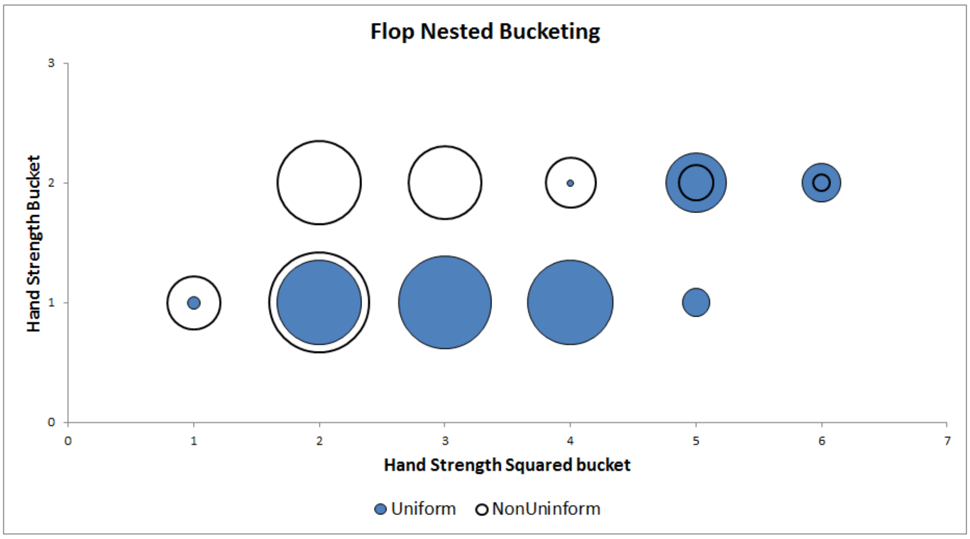

Figure 5.5: Uniform vs Non Uniform split with 8x3 nested bucketing configuration

5.2 Conclusions

As expected, as the number of the bucket increases (and thus its importance also increases) the non uniform functions grants a smoother decrease on the number of hands per bucket, both in 12 and 24 buckets, while the linear split tens to waste the first buckets, as in practice there are no such bad hands. This is also verified, albeit to a smaller extent on the Nested strategies. As we can see on figures5.2 , 5.3, the radius of the circles corresponding to the non-uniform strategies tend to decrease when we go to the higher buckets.

Another advantage of the non uniform split is the parametrization of the function, allowing for the tighter range of both $E[HS]$ and $E[HS^2]$ granting more control to the first bucket usage, and thus preventing wastes on the last buckets.

On the metric usage, as $E[HS^2]$ is usually lower than $E[HS]$ we see a bigger usage of the early buckets. This is consistent with the fact that $E[HS^2]$ treats hands differently according to their potential, thus hands with a higher $E[HS^2]$ are rarer.

As for the nested strategies, due to the fact that $E[HS]$ and $E[HS^2]$ are strongly correlated (i.e. there are no hands with a very high $E[HS]$ and a low $E[HS^2]$) some of the buckets are wasted.

These conclusions are reinforced on the Flop. The non-uniform split still tends to do better (i.e. discriminating better the best hands), despite having almost the same range as the uniform split.

On the nested strategies, the importance of the non-uniform split is also boosted by the amount of hands. It is clearer in Figures 5.4 and 5.5 that the non-uniform circles' radius decrease faster than the uniform ones.

Given these results, 2 different strategies will be used for the agent's game play: Linear and Nested, both with non uniform partitioning. While the Nested strategies tend to waste buckets (having buckets that will never contain hands), it is expected that the 2-dimensional characterization helps to better distinguish between hands, thus balancing the bucket waste.

As for the number of buckets it is clear that the more the better, as more buckets allow the agent to recognize a bigger number of different situations, though some restraint must be advised, as increasing the number of buckets leads to a growth on the number of possible information sets, which could lead to a smaller number of occurrences, thus never allowing the state's weight to converge.

Chapter 6

Poker Agents validation

In this chapter we present the results obtained by matching the develop agents against other known bots. Two type of experiments were made: Heads Up Poker, where one player plays against another in a one versus one match, and Tournament Poker, with 3 players per table. This allowed us to test the effectiveness of the opponent modelling module, and to see how the developed agents adapt to different adversaries.

Taking into account the results obtained in Chapter 5, four different agents will be tested: two Action Learners and two Outcome Learners, using 24 buckets, one of each using a Linear Non Uniform strategy (LNU) based on Hand Strength Squared and another using a Nested Non Uniform (NNU) strategy.

6.1 Opponents

Five bots, with varying strengths and weaknesses were selected to test our agents against:

- AlwaysCallBot: A bot that always calls (as the name implies). It is considered a tight passive player.
- AlwaysAllInBot: A bot that always bets all his money. This is considered a loose aggressive player.
- WHLBot: A bot developed by Nuno Passos [Pas11] that uses reinforcement learning to learn the Preflop stage. After the preflop, if it did not fold, it will always raise.
- Megabot: A bot developed by Luís Teófilo [TR]. This is a bot that can play 4 different strategies, and changes strategies when it is losing, in order to win.
- SimpleBot: A complex, but efficient bot. Contains several generic rules to play in Pre-Flop, and its post flop actions are based on the pot odds.

To better understand these bot's relative strength, they each played 100.000 games with each other, in a table with 0.1 $ big blinds, and the final bankrolls can be seen in Table 6.1. Each cell contains player A final bankroll versus player B. The table is then symmetric.

A B	AlwaysAllIn	AlwaysCall	MegaBot	SimpleBot	WHLBot
AlwaysAllIn	—	-63	-3.740	-20.776	-5.449
AlwaysCall	63	—	-2513	-20.742	-20.742
MegaBot	3.740	2513	—	-19.258	52.237
SimpleBot	20.776	20.742	19.528	—	76.198
WHLbot	5.449	2.794	-52.237	-76.198	–

Table 6.1: Results of the matches between the test bots

From Table 6.1 we can rank these bots by their expected difficulty (hardest to easiest):

1. **SimpleBot**: The hardest bot. It won against every single other bot, but its static strategy and lack of opponent modelling prevents it from maximizing its winnings against the easier bots.

2. **MegaBot**: The second hardest bot, and the only one with a non static strategy, which can be an additional problem to deal with, and make this an even harder opponent. However against SimpleBot this brings no advantages, as SimpleBot has no opponent modelling techniques. It should be noted thought that MegaBot managed to exploit better the AlwaysAllIn bot, maximizing its winnings.

3. **WHLBot**: The fact that this bot learns how to play from scratch puts him at a disadvantage against the other bots, losing to both SimpleBot and MegaBot. This can be a problem when matched against our agents, as none of the players will know how to play, thus the initial developed strategies can be completely wrong and thus obtain unpredictable outcomes.

4. **AlwaysCall**: The second easiest, having won only against the AlwaysAllIn Bot, and by a very small margin.

5. **AlwaysAllIn** : It is expected to be the easiest bot to win against, having been beaten by every other bot.

6.2 Heads Up Poker

In every experiment, each agent played 200.000 games versus each opponent, with no previous knowledge, to test the agent's learning technique. The results presented in the plots are a moving average of the amount of big blinds won (or lost) for the past 1000 games.

Another aspect worth mentioning about the experiments is that, in order to reduce variance, each set of games is actually played twice (on a 2 player match). This means that after the first 100.000 games are done, the agent plays the same 100.000 games again, this time with the initial positions swapped, meaning our agent is getting the cards our opponent had last time. The rationale behind this change is simple. If on the first time I won the game (because I had better cards) it is expected that I lose the second time (my opponent now has the better cards). Therefore

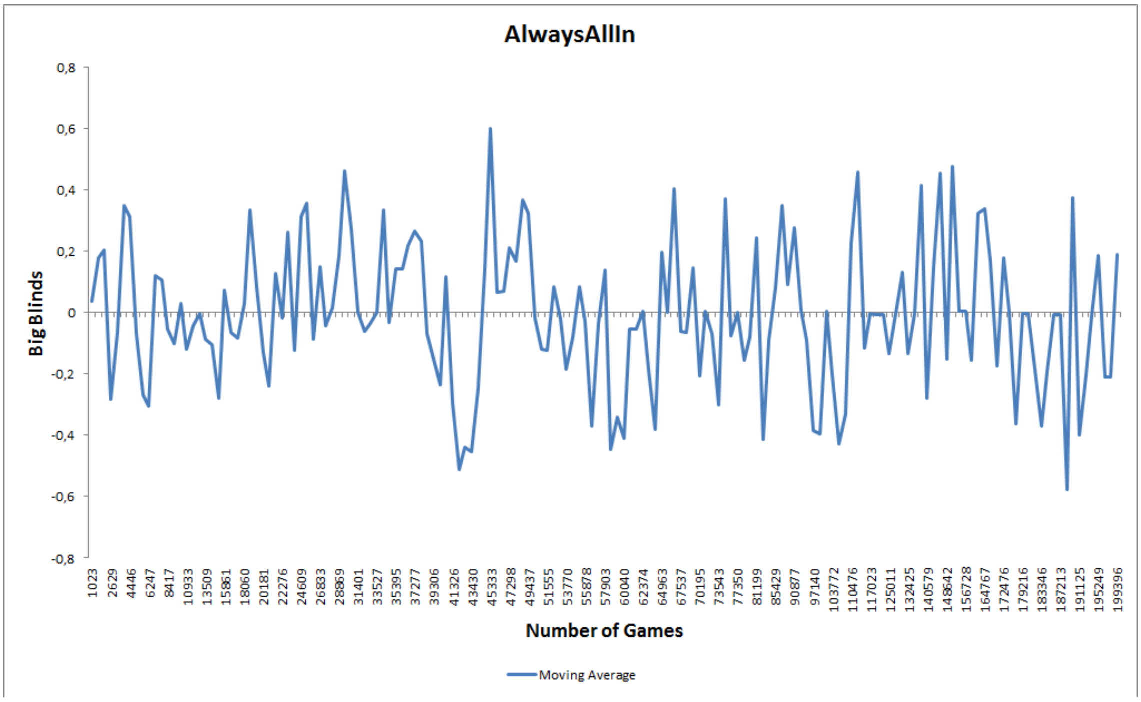

Figure 6.1: Outcome Learner (LNU) vs AlwaysAllIn bot

only after this seat permutation we can tell who is the winner, as we dealt the same cards to both players.

Each game has a big blind of 0.1, and the results will be presented in big blinds per 1000 matches, as the big blinds set the tone for the betting rounds. So if a player wins 10 big blinds per 1000 games, we can expect it to win 1 dollar per 1000 matches in these experiments «.

6.2.1 Outcome Learner

Due to this agent's architecture and learning methodology, it is expected that its results are not stellar. While it should be capable of winning against a player with no strategy, it is not expected that this agent wins by much against the harder adversaries.

6.2.1.1 Outcome Learner with Linear Non Uniform strategy

In this subsection the results obtained from playing an Outcome Learner (with a Linear Non Uniform strategy) versus the 5 opponents are shown.

Versus AlwaysAllIn bot

As can be seen in Figure 6.1, while the agent managed to win some games, it couldn't develop a strategy to consistently win against this bot, although it did manage to minimize his losses. Final Bankroll: -179$.

Versus AlwaysCall bot

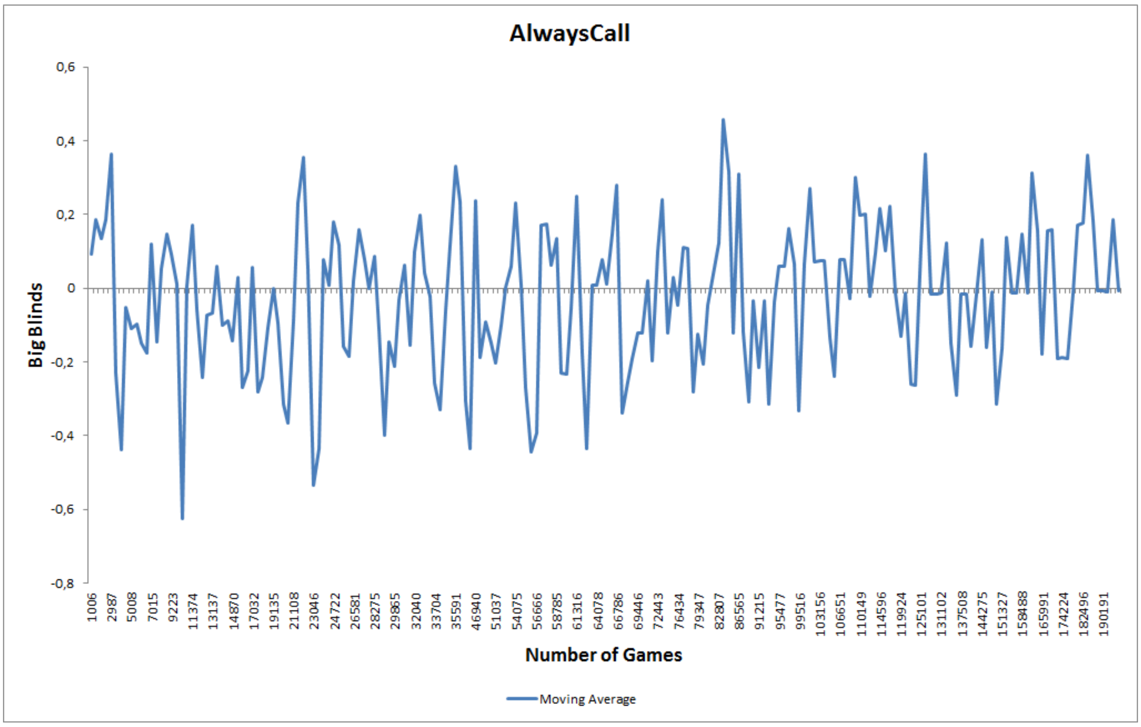

Figure 6.2: Outcome Learner (LNU) vs AlwaysCall bot

Against this bot the results are slightly better. While our agent lost, Figure 6.2 suggests a slow convergence to better values (the negative spikes are decreasing), and perhaps given more games against this bot our agent would be able to develop a winning strategy. Final Bankroll: -98.176$.

Versus WHLBot

This is an interesting match up, as both players start the game without knowledge of how to play it, and are both learning with the results. As can be seen in Figure 6.3 our agent managed to reach a positive average winning soon, and it won more than it lost, though it was not capable of exploiting the WHL bot post flop weaknesses to their full potential. Final Bankroll: 14.595$

Versus MegaBot

This was expected to be a tough opponent and these results confirm it. As can be seen in figure 6.4 our agent had very few streaks with a positive average, and ended up losing most of the matches. Final Bankroll:-25.272$

Versus SimpleBot

After these previous results, we can expect our agent to lose against this SimpleBot. Figure 6.5 shows that our agent lost, and by more (on average) than against MegaBot. Final Bankroll: -103.907$

6.2.1.2 Outcome Learner with Nested Non Uniform strategy

After seeing the results obtained by the Outcome Learner with a Linear Non Uniform strategy, the bucketing strategy was changed to a Nested Non Uniform and the experiments were repeated.

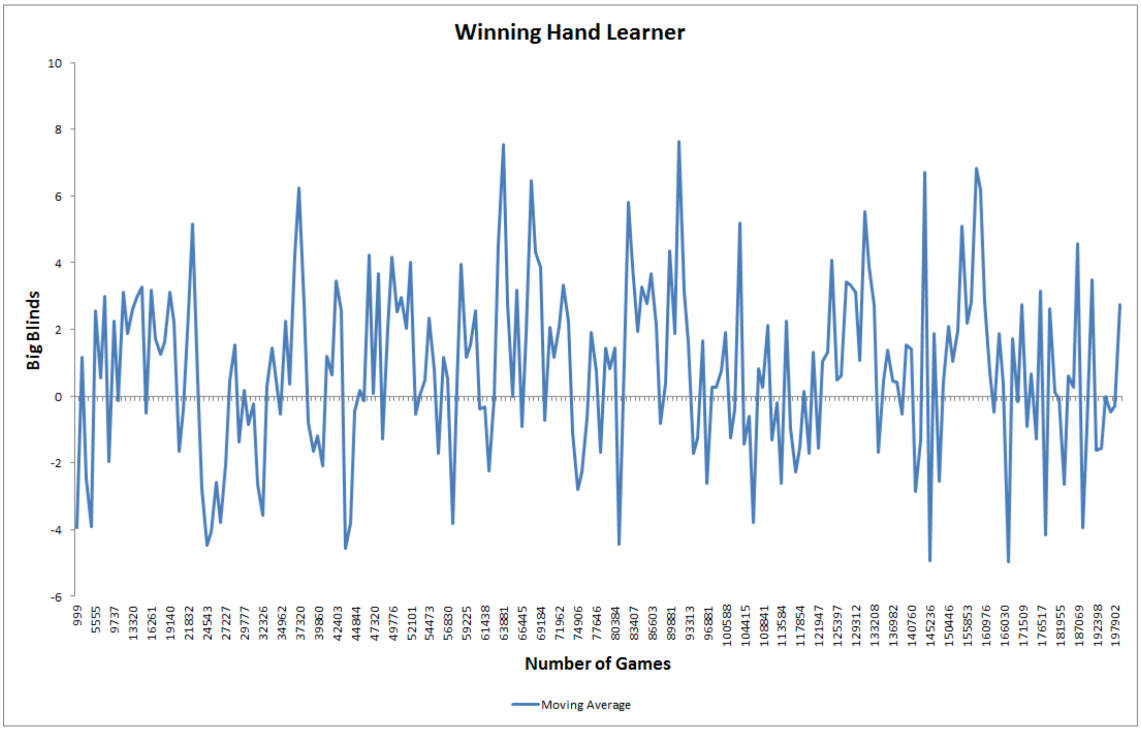

Figure 6.3: Outcome Learner (LNU) vs WHLBot

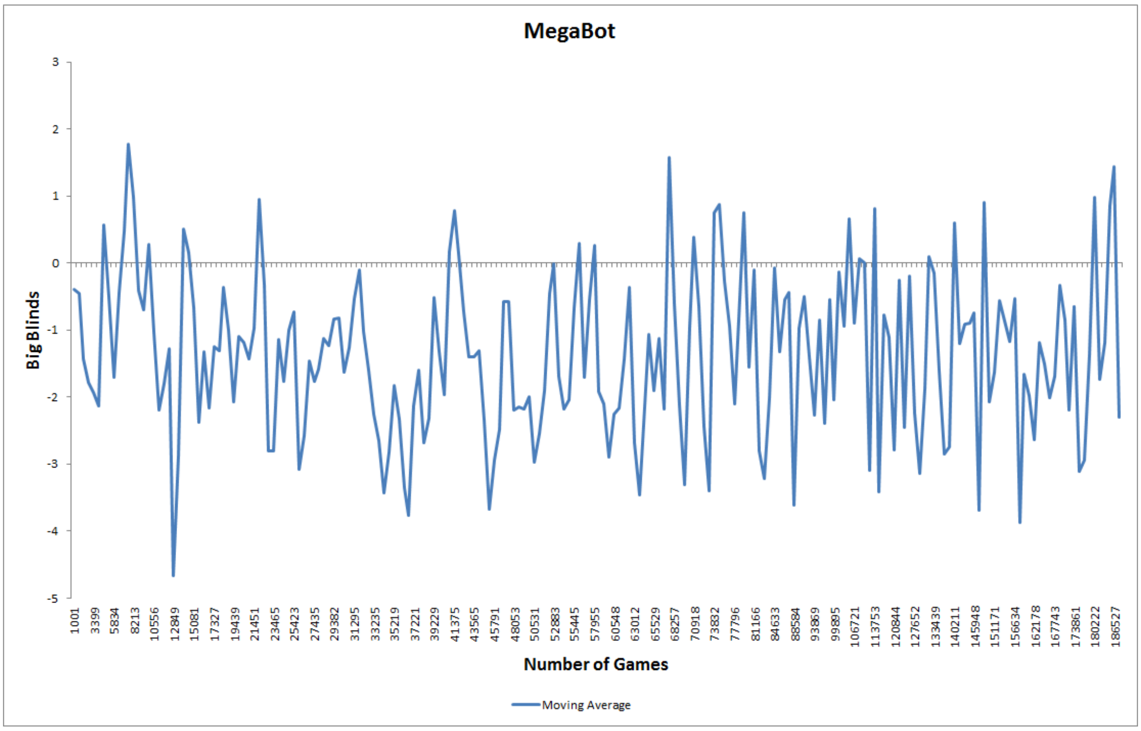

Figure 6.4: Outcome Learner (LNU) vs MegaBot

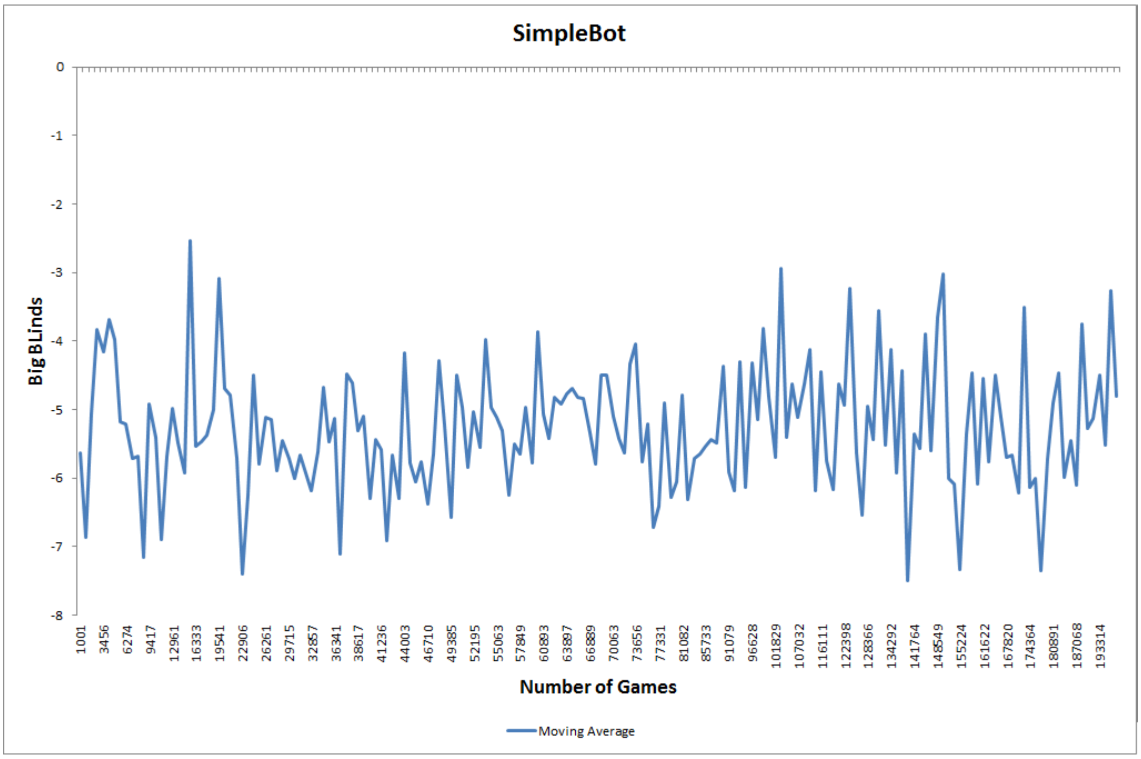

Figure 6.5: Outcome Learner(LNU) vs SimpleBot

Versus AlwaysAllIn bot

As with the linear bucketing strategy, our agent only managed to develop a strategy to minimize losses (as can be seen in Figure 6.6 and could not fully exploit this bot's weaknesses. The change in positive and negative average suggest a non-converging strategy being developed. Final Bankroll: -599$

Versus AlwaysCall bot

As when the linear strategy was employed, Figure 6.7 suggest an improvement of our agent's strategy, as the number (and amount) of positive average increases, though this improvement was clearer when using a linear bucketing strategy. Final Bankroll: -638$

Versus WHLBot

As can be seen in Figure 6.8 the nested strategy produces worse results than the linear strategy. Again, a clear convergence cannot be seen. Final Bankroll: -3.498$

Versus MegaBot

Figure 6.9 suggests that the nested strategy worked better against this opponent. While our agent still lost, it managed to keep its losses smaller. But again, there is no clear convergence to a winning strategy. Final Bankroll: -15.119$

Versus SimpleBot

Against this bot, changing the bucketing strategy produced no improvements. Figure 6.10 shows the same kind of losses as shown in Figure 6.5, and there is no clear improvement.

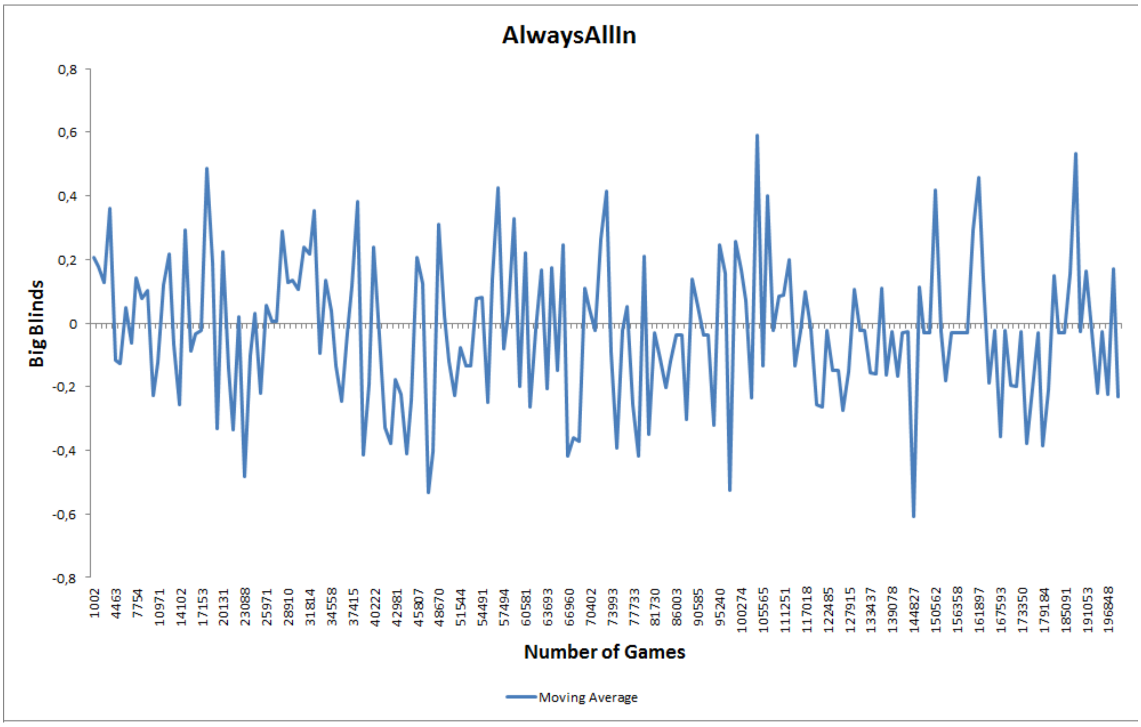

Figure 6.6: Outcome Learner(NNU)vs AlwaysAllIn

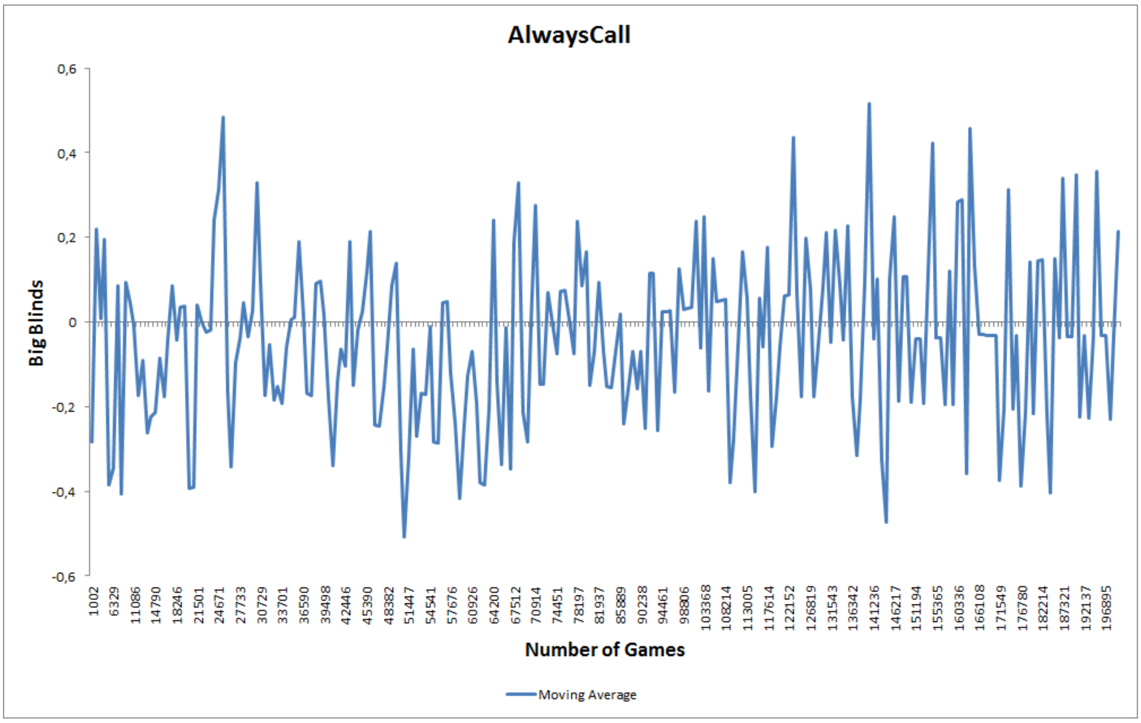

Figure 6.7: Outcome Learner(NNU) vs AlwaysCall

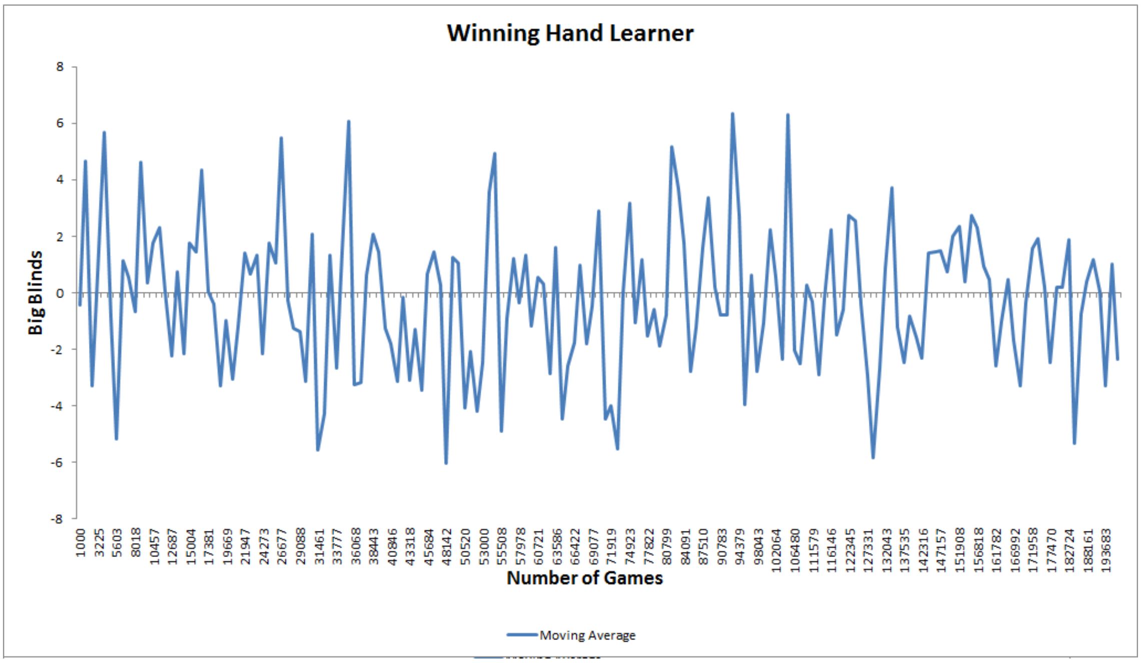

Figure 6.8: Outcome Learner(NNU) vs WHLBot

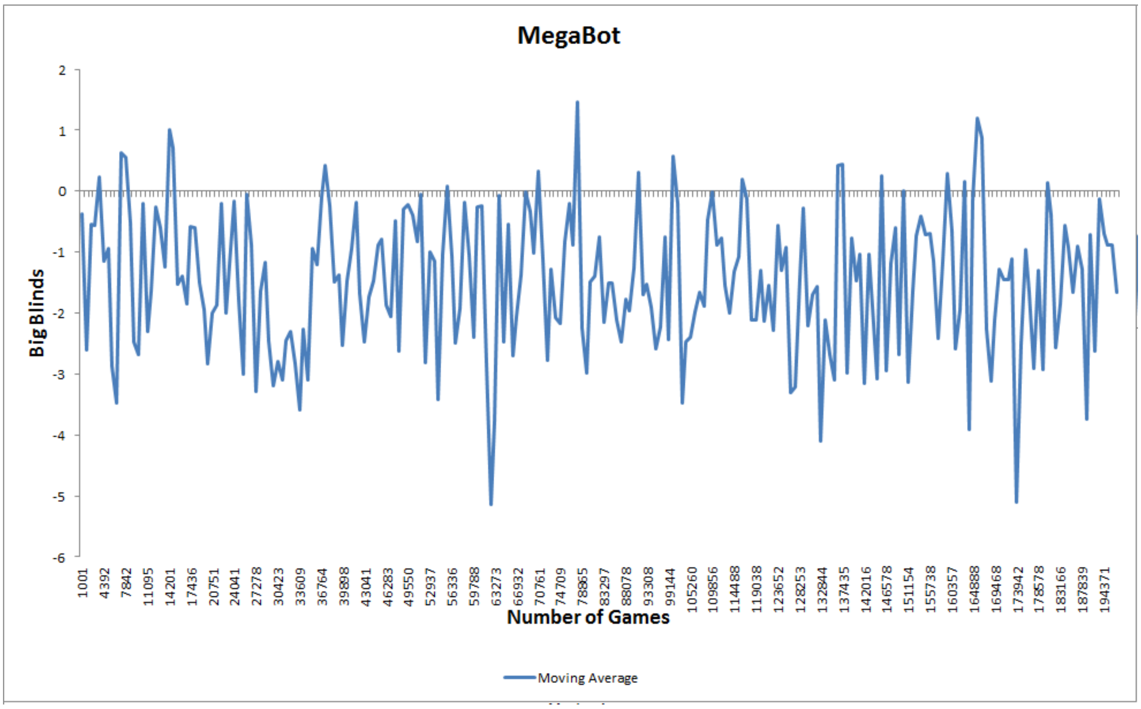

Figure 6.9: Outcome Learner(NNU) vs MegaBot

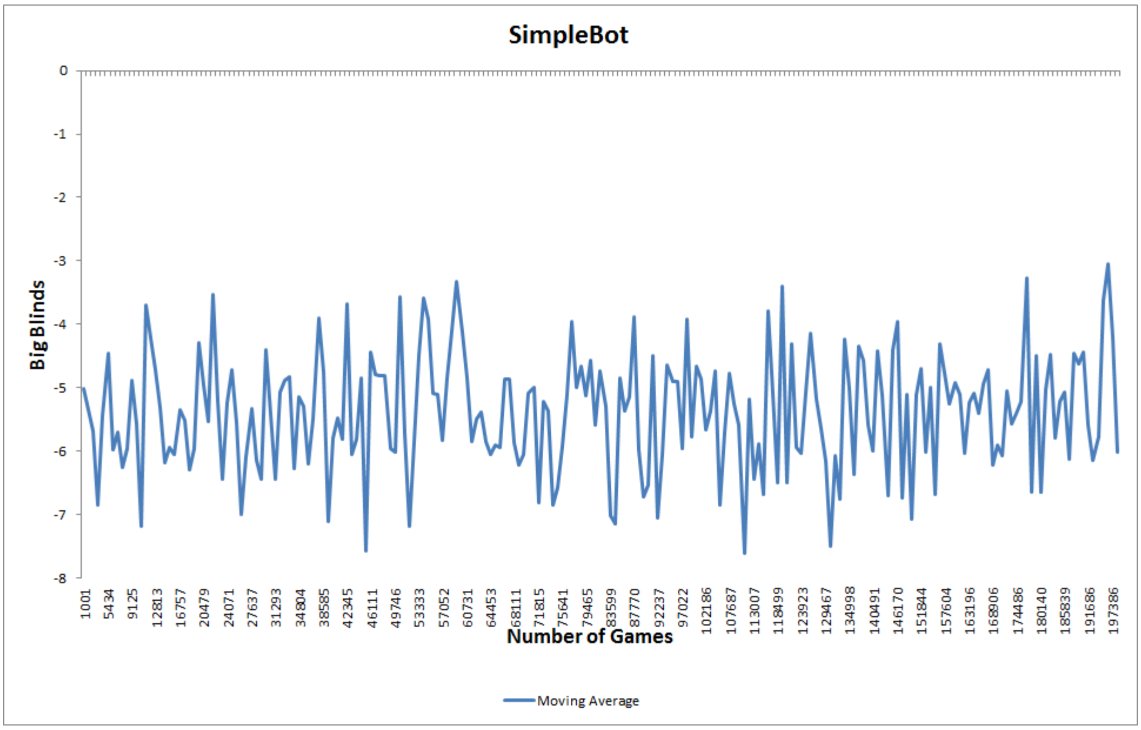

Figure 6.10: Outcome Learner(NNU) vs SimpleBot

6.2.2 Action Learner

This agent's development was more focused on the game of poker itself, and thus we expect better results than those obtained with the Outcome Learner.

6.2.2.1 Action Learner with Linear Non Uniform strategy

In this subsection the results obtained from playing an Action Learner (with a Linear Non Uniform strategy) versus the 5 opponents are shown.

Versus AlwaysAllIn bot

As can be seen in figure 6.11 the agent started losing but around the 2000th game the average was already positive and it only increased. There is a clear convergence to a winning strategy that wins between 5 and 7 big blinds every 1000 games, on average. Final bankroll: 92.386$. It must be noted that this final bankroll is higher than both the one obtained by SimpleBot and Megabot, when playing the initial ranking games (whose results are shown in Section 6.1.

Versus AlwaysCallBot

After the previous results, we could expect nothing but a solid win and as can be seen in Figure 6.12 the moving average for the first 1000 games is already positive, and it tends to increase with the number of games. As expected, these numbers are smaller than when playing against the AlwaysAllIn bot. Final bankroll: 91.957$. Again, this is a very high bankroll, higher than any other bot obtained before.

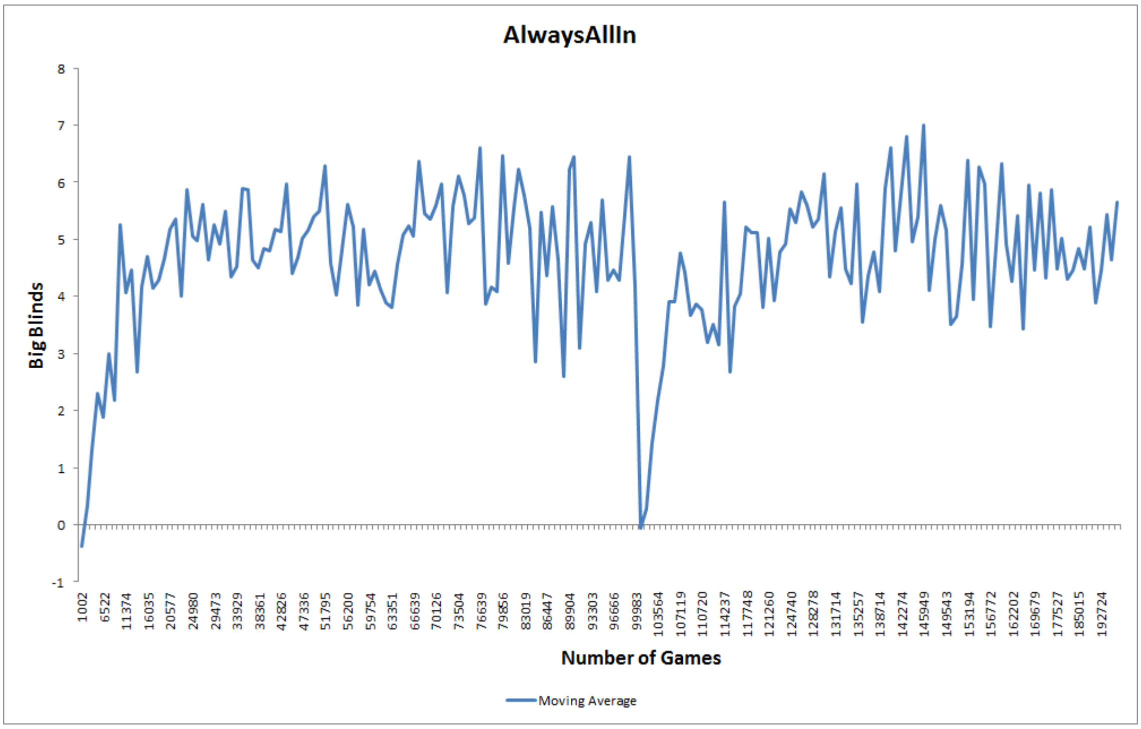

Figure 6.11: Action Learner (LNU) vs AlwaysAllIn

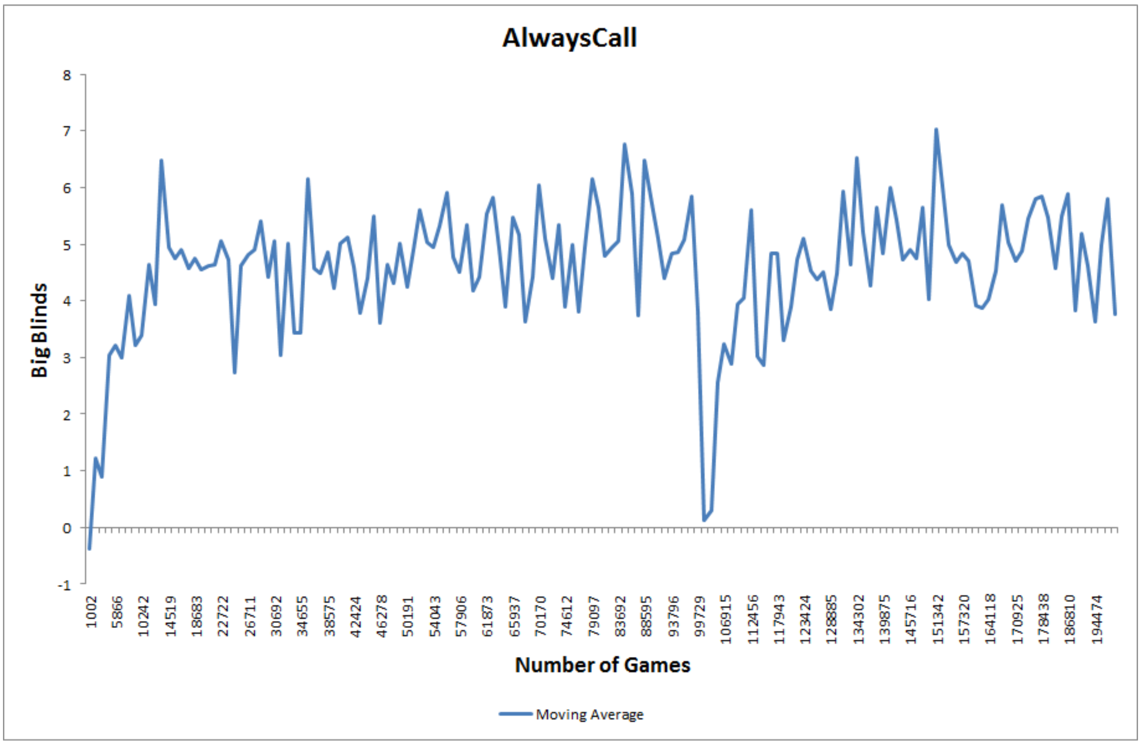

Figure 6.12: Action Learner (LNU) vs AlwaysCall

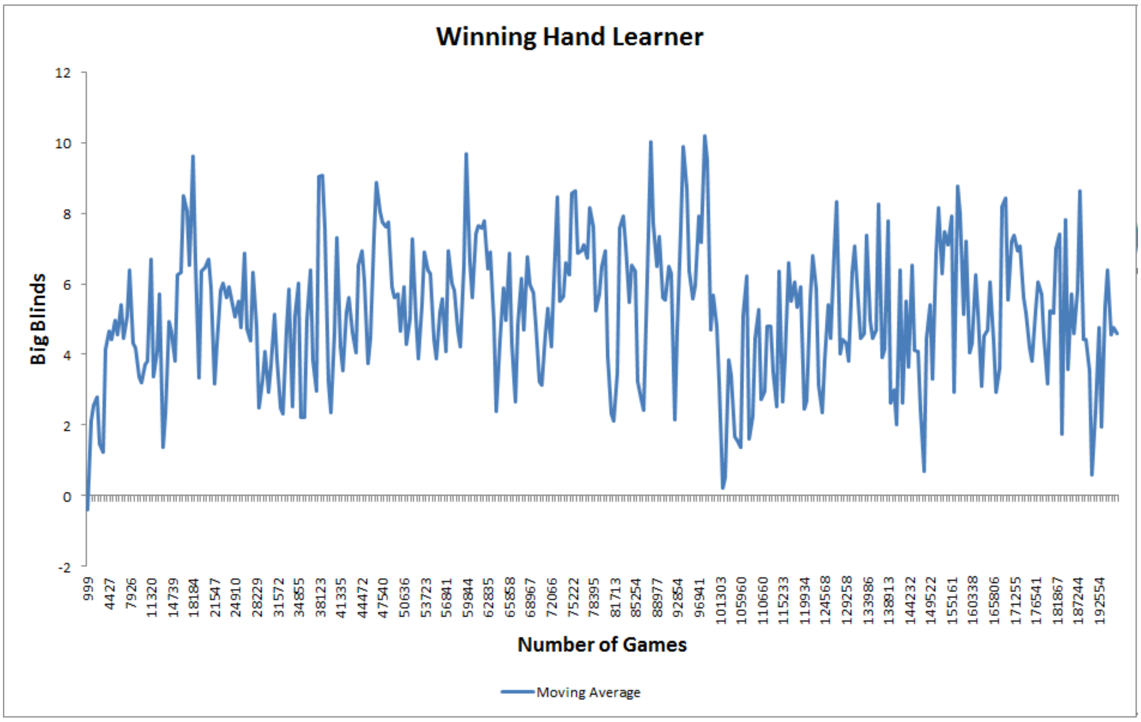

Figure 6.13: Action Learner (LNU)vs WHLBot

Versus WHLBot

Playing against this bot may prove a challenge, due to their learning phases. As can be seen in Figure 6.13, our agent bested the WHLBot since the beginning, ending up exploiting the always all in phase. It should be noted though that there is a higher variance on the expected winnings (from 2 big blinds to 10). This is a symptom of both players adapting to each other's strategy, and our agent ends up winning, exploiting the lack of post flop strategy in the WHLBot. Final Bankroll: 101.087$

Versus Megabot

As this is one of the toughest opponents, due to its non-static strategy, the expected winnings are much smaller. As can be seen in Figure 6.14 the agent started losing (as expected) and it takes our agent a while to realize the change in the opponent's strategy, so the average winnings are always oscillating between positive and negative. Final Bankroll: 6.280$

Versus SimpleBot

This is the hardest opponent, and as can be seen in Figure 6.15 the agent started losing but the average winnings are slowly increasing, and the graphic suggest that it could still improve more, if more games are allowed. Final Bankroll: 2.040$

6.2.2.2 Action Learner with Nested Non Uniform strategy

This subsection shows the results of playing an Action Learner with a Nested Non Uniform strategy against the same opponents.

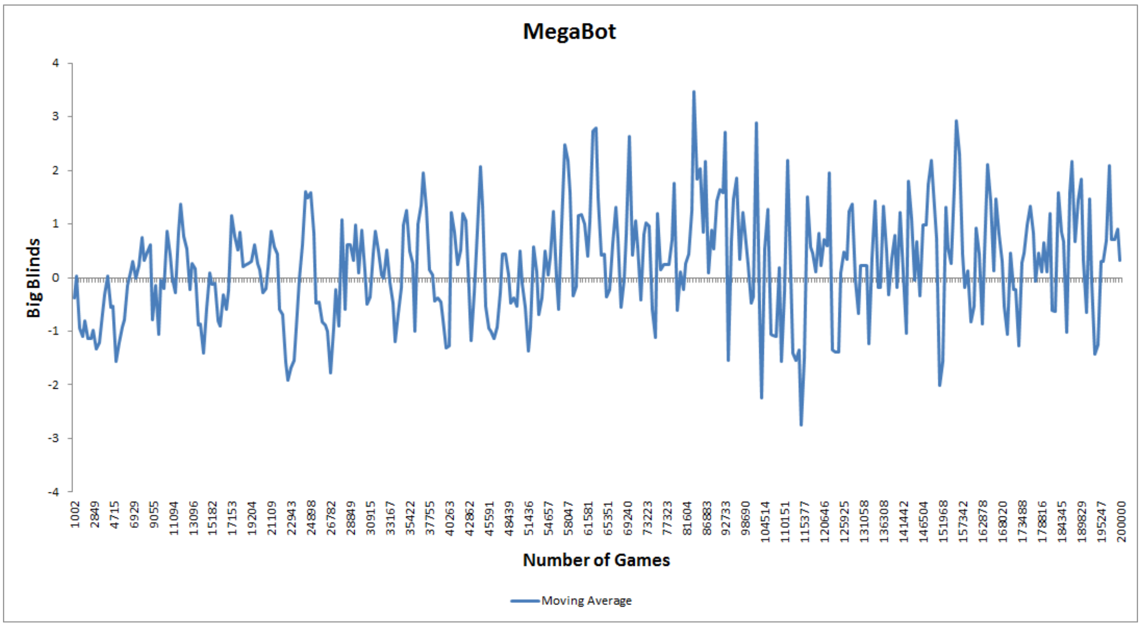

Figure 6.14: Action Learner (LNU) MegaBot

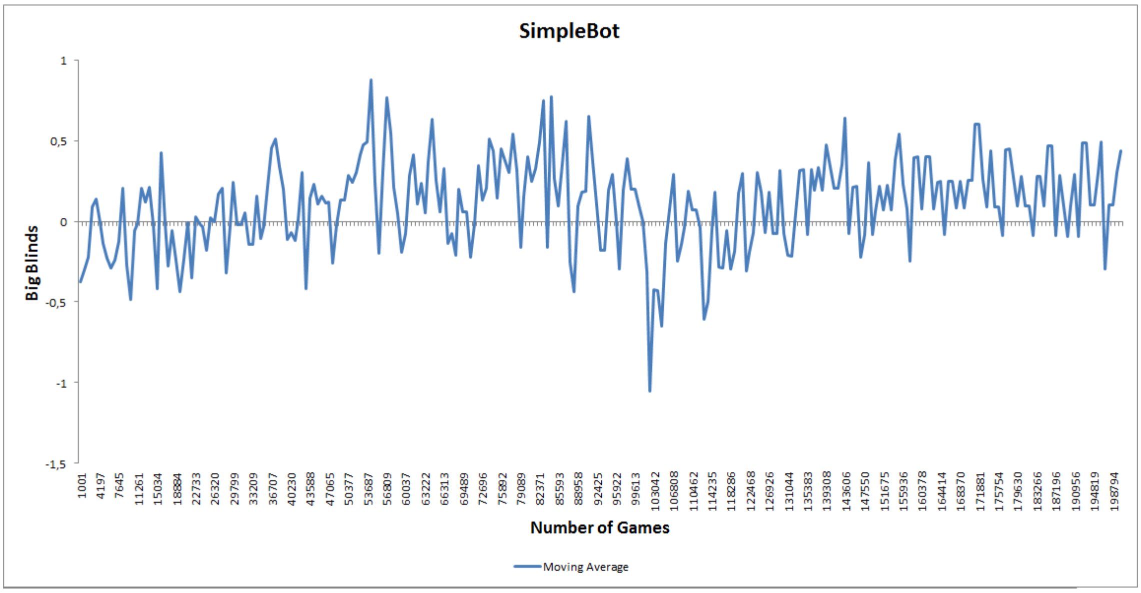

Figure 6.15: Action Learner vs SimpleBot

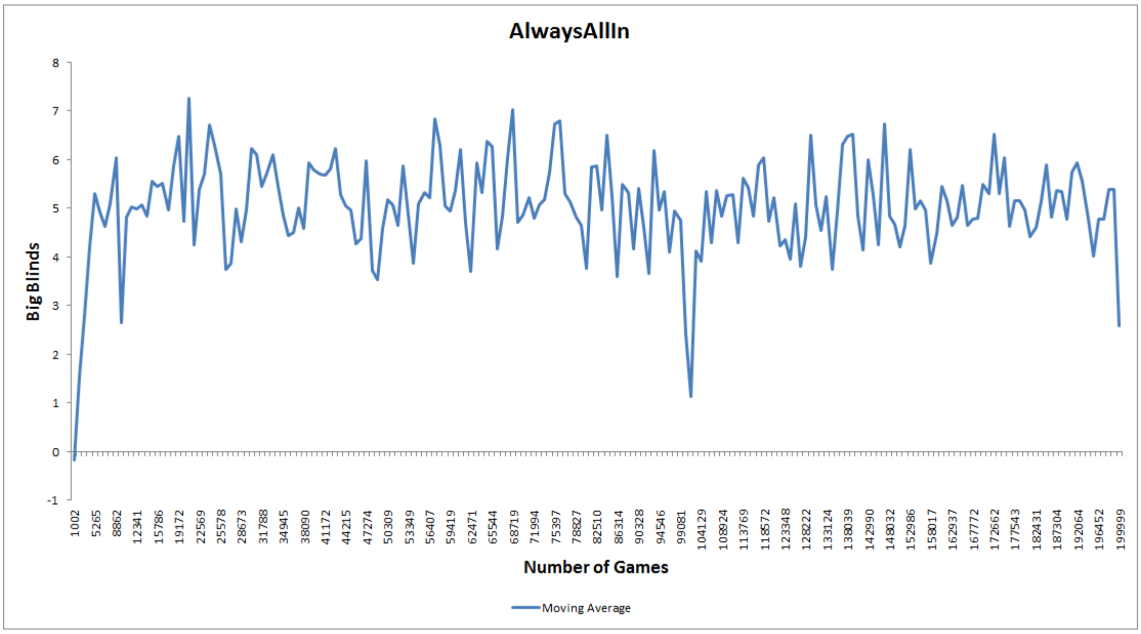

Figure 6.16: Action Learner (NNU) vs AlwaysAllIn -

Versus AlwaysAllIn bot

Again, as expected, the agent soon learnt how to beat this simple opponent, as can be seen in Figure 6.16, but achieving worse results that its linear counterpart, where we could foresee a bigger improvement of the winnings. Final Bankroll: 99.280$

Versus AlwaysCall bot

With this strategy we expected similar results as in Figure 6.12, and that was what we obtained, as can be seen in Figure 6.17, with the same average winnings. Final Bankroll: 99.596

Versus WHLBot

The initial games against this adversary are somewhat random, as both agents are trying to learn how to play the game, but as can be seen in Figure 6.18, as the number of games increases, our agent's average winnings kept increasing. The high variance can be attributed again to both players adjusting their play style. Final Bankroll: 85.790$.

Versus MegaBot

Matches against this adversary are always interesting because when it changes strategies, the agent has to learn how to play against a "new" adversary. This can be seen in Figure 6.19, with the erratic oscillation of the winnings. Nevertheless, our agent still managed to obtain a final positive bankroll. Final Bankroll: 8.940$.

Versus SimpleBot

Again SimpleBot proved to be the hardest of the opponents, with the average winnings staying relatively low, and oscillating between positive and negative, though there is a clear positive tendency. Final Bankroll: 2.760$.

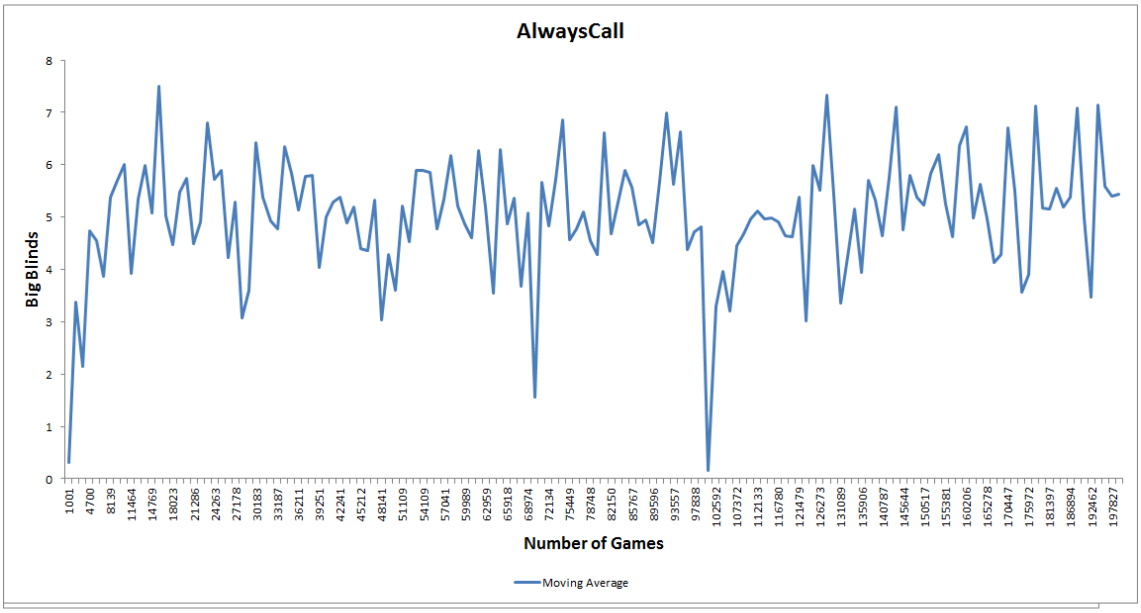

Figure 6.17: Action Learner (NNU) vs AlwaysCall

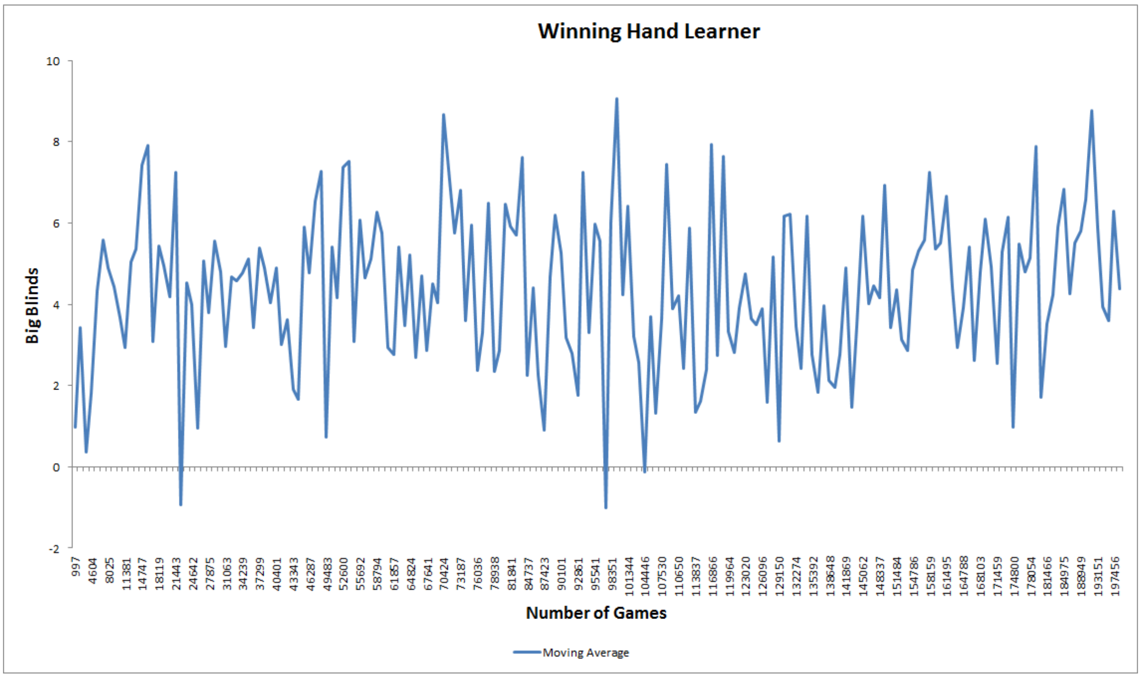

Figure 6.18: Action Learner (NNU) vs WHLBot

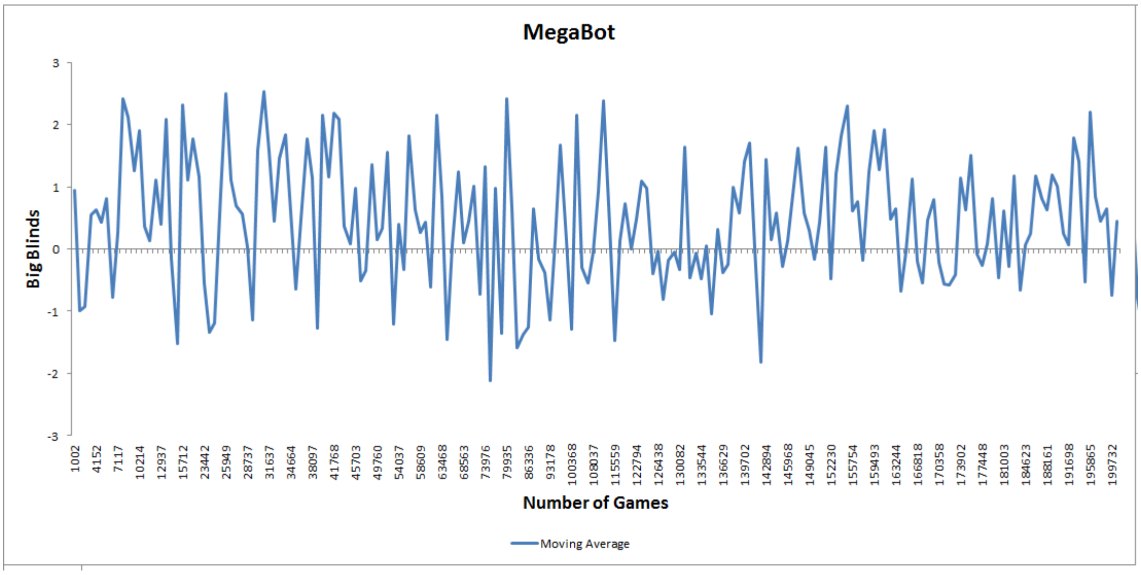

Figure 6.19: Action Learner (NNU) vs MegaBot

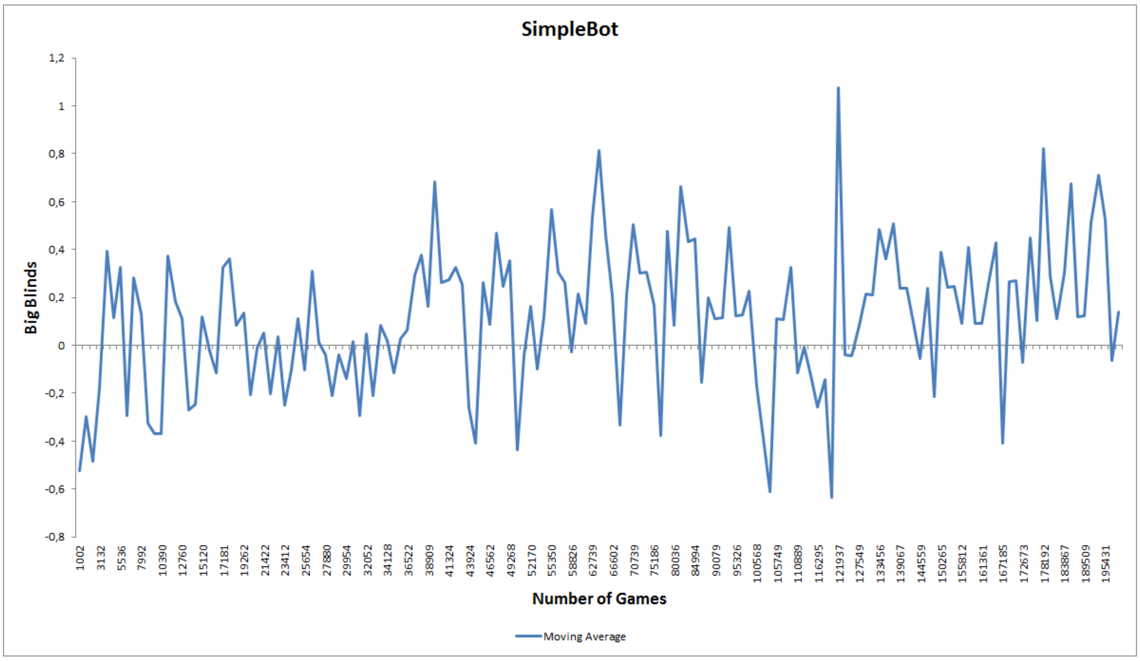

Figure 6.20: Action Learner (NNU) vs SimpleBot

6.2.3 Analysis

The first conclusion to draw from these Head's Up games is that while we did not expect the Outcome Leaner to perform as well as the Action Learner, we still expected some wins against the easier opponents. While this was not verified, there are some cases that suggest a later convergence, thus perhaps more games needed to be played.

The second conclusion is that there is no big difference between a linear strategy and a nested one. There were some trade-offs when choosing one over the other, but the end results were similar.

Another conclusion we can draw is that the Action Learner can develop good strategies against the chosen opponents, and even adapt to changes in strategies, as was observed when playing against MegaBot and WHLBot.

6.3 Tournament Poker

Given the results obtained in 6.2, we selected the Action Learner with a Linear Non-Uniform strategy to test the learning techniques with more than one opponent. While it is expected that the agent will reproduce the results obtained playing Head's On Poker, this is not an assured thing, as winning strategies for both kinds of games are very different.

For the games, three different set ups were created, in order to test our agent in different situation. MegaBot was excluded from these scenarios due to time constraints, as it is the slowest of the agents. On each table were played 80.000 games with seat permutations, totalling 480.000 games per table.

The only change made to the implementation was on the calculation of each hand's strength, which had to be accommodated for an increase in the number of players.

6.3.1 Versus Two Simple Bots

In this table we placed our agent against two SimpleBots. This is expected to be the hardest situation, as our agent is up against two versions of the hardest opponent. As can be seen in Figure 6.21, while the average winnings stabilizes at a negative value, it is a very small one. While it is not clear, Figure 6.21 could also suggest that as the number of games increases, our agent would eventually converge to a winning strategy, given that after the 200.000 games mark we see some positives averages.

6.3.2 Versus Two AlwaysCall bots

In this table we placed our agent against two AlwaysCall bots. This is expected to be one of the easiest situations, as our agent is up against two versions of an opponent it could already soundly beat before. Plus, given the nature of the AlwaysCallBot, we can ensure that every game will end in the showdown, which is perfect for a learning agent, as every action can be rightfully evaluated as good or bad (there is no longer uncertainty if a fold was a correct option). As can be seen

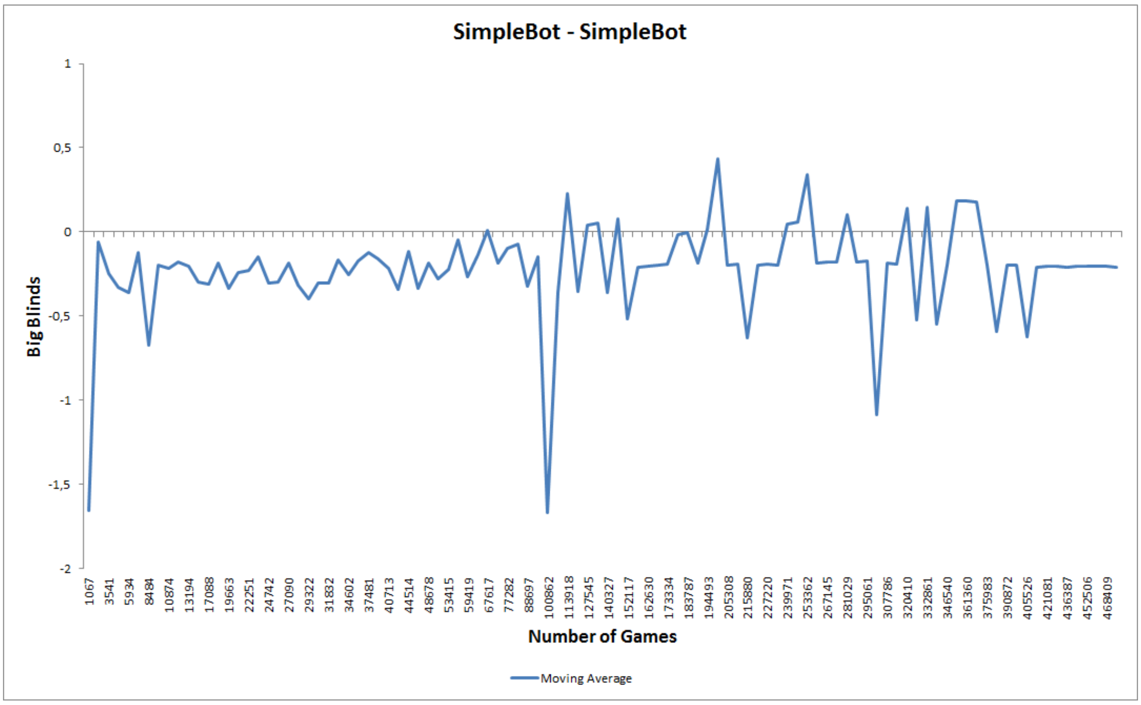

Figure 6.21: Action Learner in a table with two SimpleBots

in Figure 6.22, our agent quickly develops a strategy to defeat these opponents and by the larger margin yet (around 20-30 big blinds per 1000 games). The sudden drops in winnings could be related with the learning methodology, which puts much emphasis on regret minimization, thus after an unlucky streak our agent becomes careful, hence the spikes.

6.3.3 Versus a SimpleBot and a AlwaysCall bot

In this table we placed our agent against one SimpleBot and one AlwaysCall bot. As there is an AlwaysCall bot we have again the assurance of a showdown, which helps our agent, thus it is expected that it can win in these situations. Another important factor to consider in this table is that these are two different agents, with different play styles, thus opponent modelling has a very big impact on the results. As can be seen in Figure 6.23 the average winnings are positive, though Figure 6.24 shows that both the SimpleBot and our agent managed to stay at a positive bankroll, exploiting the AlwaysCall bot. Another interesting point that can be seen in Figure 6.24 is that our agent's bankroll is bigger than the SimpleBot's, suggesting that either we are winning more showdowns than him, or we are exploiting the AlwaysCall bot better.

6.3.4 Conclusion

Playing poker with more than one opponent is an entirely different game than playing Head's Up Poker. It adds much more complexity to the game, and magnifies the importance of opponent modelling.

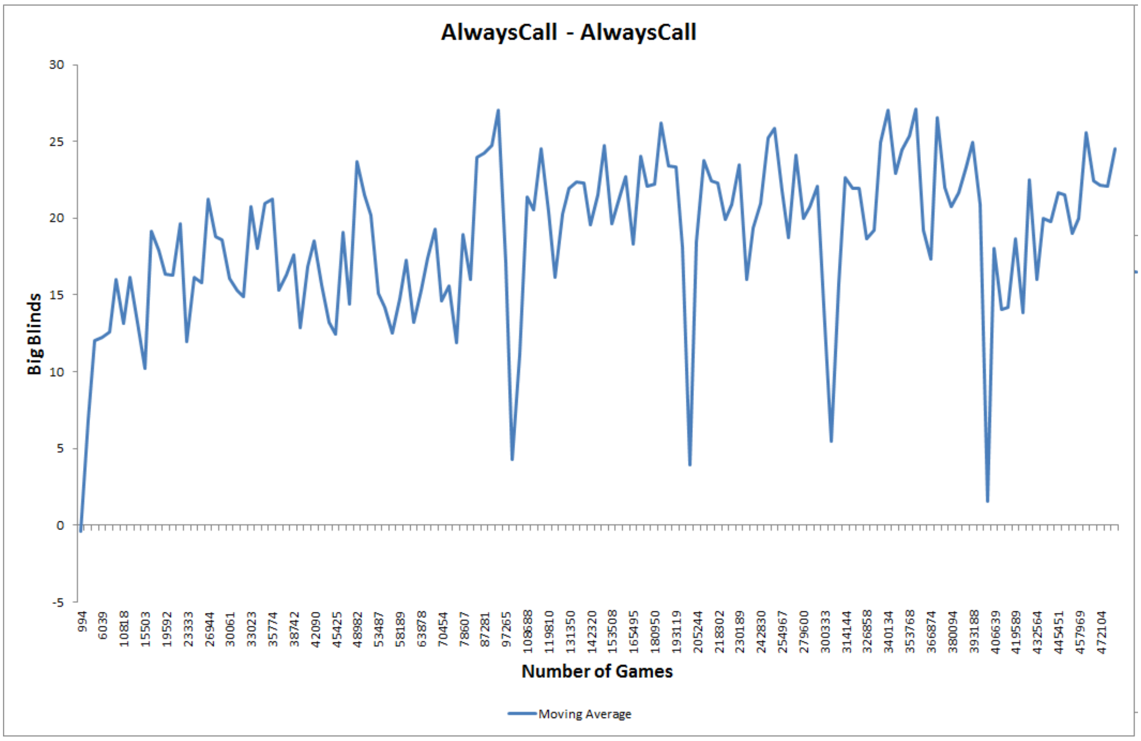

Figure 6.22: Action Learner in a table with two AlwaysCallBot

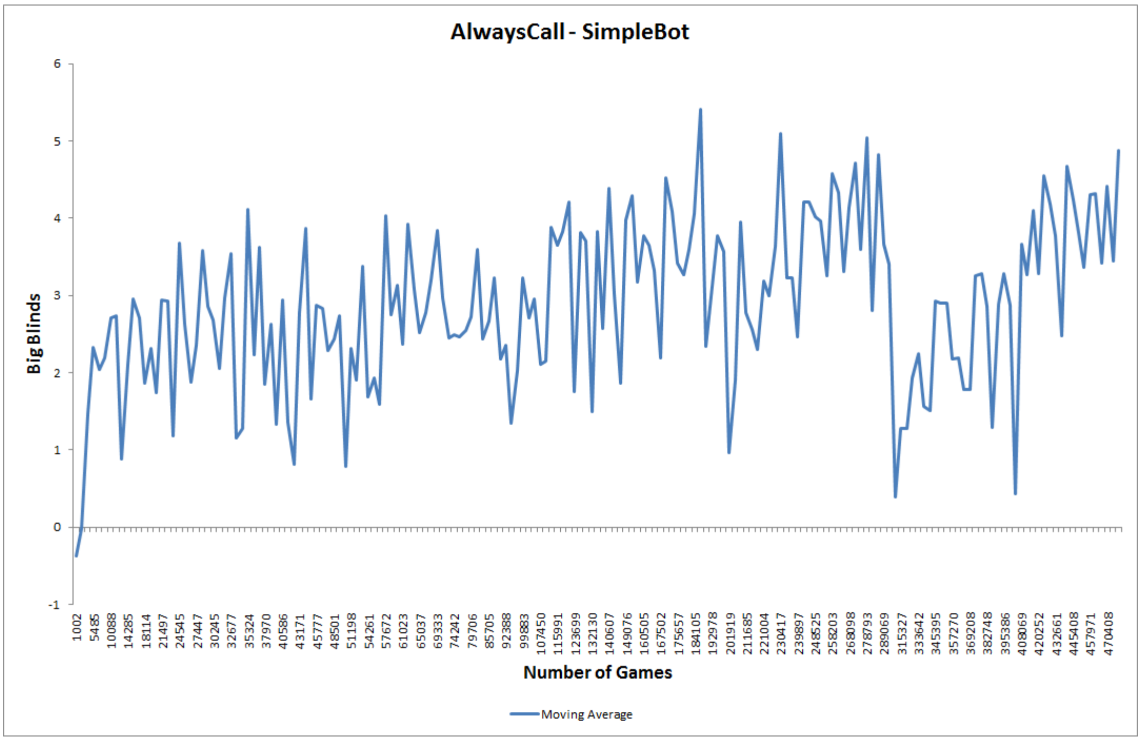

Figure 6.23: Action Learner in a table with one SimpleBot and one AlwaysCall bot

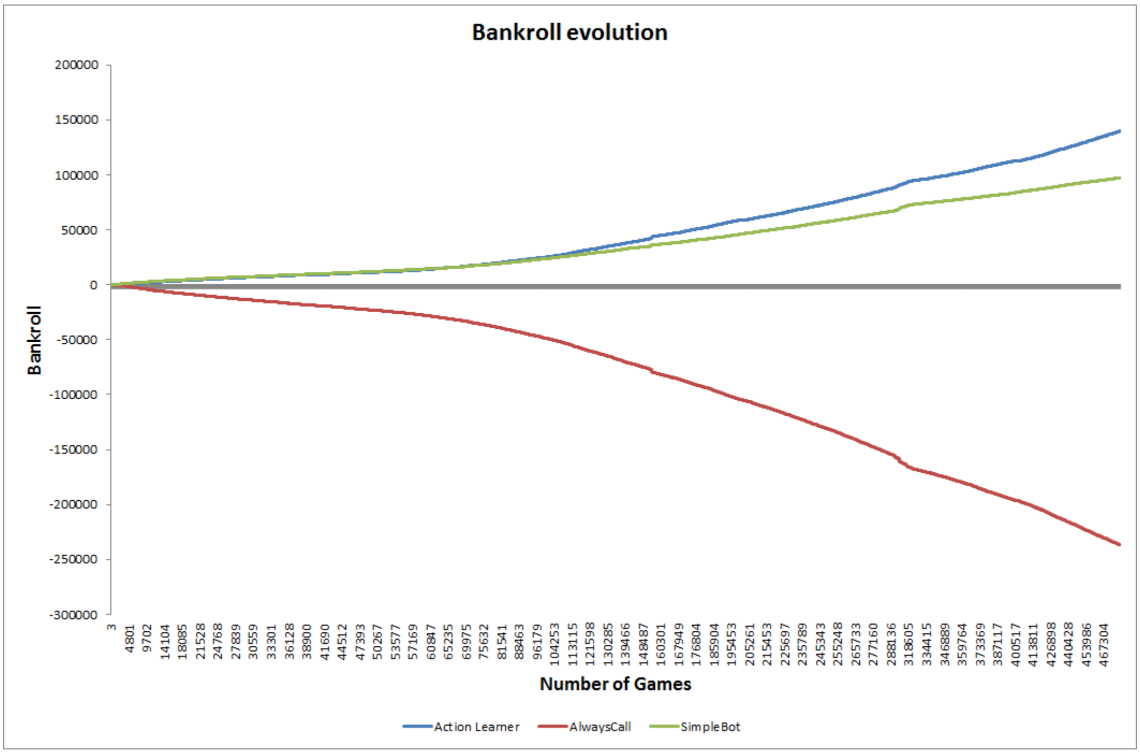

Figure 6.24: Bankroll for table 3 - One AlwaysCall bot and one SimpleBot

One problem of experimenting on multi-player tables is the sheer number of necessary games, due to the players permutations ($N!$). If we have a table with 3 players we have $3! = 6$ permutations, while 4 players introduces $4! = 24$ permutations. Thus, due to computational reasons, all experiments were made with 3 players - our agent plus two adversaries.

The three chosen configurations allowed us to test three distinct situations:

- Two competent adversaries. In Section 6.3.1 our agent had two face two opponents that play the game with some strategy, the closest to a real world situation as possible. While our agent was not able to win, it did manage to minimize his losses, and had more games been allowed, it would probably had converged to a winning strategy.
- Two easy adversaries. In Section 6.3.1 our agent had two face two opponents that have no strategy at all. This allowed us to observe how our agent's learning methodology worked in a perfect learning scenario. As could be observed, convergence to a winning strategy was fast.
- One easy and one hard adversary. This is the intermediate table, which allowed us to observe how well our agent would do against an exploitable adversary (AlwaysCall bot) in the presence of another good player (SimpleBot). As could be observed in Figure 6.23 our agent was able to exploit the AlwaysCall bot better than SimpleBot was, due to the opponent modelling capabilities, thus ending up with a higher bankroll.

Chapter 7

Conclusions

This chapter presents the conclusions of this research, through confrontation against the defined goals. It also presents future work in this domain.

7.1 Goal achievement

The main purpose of this work was to create an agent that learnt how to play Poker from scratch, based on Reinforcement Learning techniques. A secondary goal was to make this approach as general as possible, in order to be able to port it to another game or even another domain. Another important goal was to test the impact of a good abstraction in the play style of a poker agent.

The main goal was reached, with the Action Learner being able to beat every opponent, some by a very large margin, and even performing reasonably in a multi-player scenario. While these results are good, it is also important to point out that the agent converged to a good strategy rather quickly (less than 10.000 games) on most cases, when comparing to the current best algorithm, Counterfactual Regret Minimization (CRF) that takes around 2.000.000 games to converge and around 2 weeks [Joh07]. Another key result is the agent's adaptability to different strategies, as seen when playing against non-static players. Its inherent learning capabilities made it naturally suited to deal with non-static strategies, treating each "new" strategy as a new learning procedure. One good surprise was the capability to find good exploitative strategies against the simpler bots, maximizing its winnings.

Regarding the second goal, an attempt was made with the Outcome Learner to create a more general approach to the game. While this approach did reasonably well against the most trivial opponents, it lost by a big margin against opponents that applied some kind of strategy. There was some indication of convergence to a better strategy, but an unreasonably large amount of games would have to be played. This is not unexpected, as a more general approach loses (by definition) details, and in games as complex as Poker, these details matter.

As for the abstraction, four different bucketing strategies were analysed, concluding that between Uniform and Non-Uniform, the latter was always better, as it allows a better usage of the existing buckets. This lead to testing two strategies, Linear and Nested. While the Linear provided an overall better usage of the buckets (as it had no empty buckets), the Nested strategy characterized hands better, looking both at strength and potential, and it was not clear before the tests which one was better. During the experiments we found no real differences when employing either strategy, which implies that these differences were not as big as we perceived, and were nullified by the amount of buckets used, and the amount of games we played.

7.2 Future work

This work can now proceed in two distinct directions: one, we can try to improve the Outcome Learner, or try to develop another agent, whose implementation is more general than the Action Learner, and therefore more portable to other games/domains. Another direction is improving on the Action Learner, in order to create an even better agent that applies Reinforcement Learning to learn the Poker Game. One very practical way of improving this agent is by adding Abstractions to the actions. The raise action is very complex in itself, and can be split into more actions, such as "Raise half pot", "Raise full pot", "Raise twice the pot". This allows for a broader strategy and better overall game play than a single raise action, as it is currently implemented.

For both agents, a better opponent modelling technique might be implemented, allowing these agents to distinguish between more players, creating more (and better) counter-strategies.

References

[BB03] D Billings and N Burch. Approximating game-theoretic optimal strategies for full-scale poker. *INTERNATIONAL JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE*, pages 661–668, 2003.

[BDS04] D Billings, A Davidson, and T Schauenberg. Game-tree search with adaptation in stochastic imperfect-information games. *Computer and Games*, pages 21–34, 2004.

[BDSS02] D Billings, A Davidson, J Schaeffer, and D Szafron. The challenge of poker. *Artificial Intelligence*, 134(1-2):201–240, 2002.

[Bil95] Darse Billings. *Computer Poker.* Master thesis, University of Alberta, 1995.

[Bil06] D Billings. *Algorithms and assessment in computer poker.* PhD thesis, University of Alberta, 2006.

[BJ08] Michael Bowling and Michael Johanson. Strategy evaluation in extensive games with importance sampling. *Proceedings of the 25th international conference on Machine learning*, pages 72–79, 2008.

[BPSS98] Darse Billings, Denis Papp, Jonathan Schaeffer, and Duane Szafron. Opponent modeling in poker. *PROCEEDINGS OF THE . . .*, pages 1–7, 1998.

[BSS99] Darse Billings, Jonathan Schaeffer, and Duane Szafron. Using probabilistic knowledge and simulation to play poker. *AAAI NATIONAL CONFERENCE*, pages 697–703, 1999.

[CEW98] Greg Coman, B Evans, and Rob Wootton. Responsible Gambling: A Future Winner. *Proceedings of the 8th National Association for Gambling Studies Conference*, 1998.

[CF08] João Carlos and Leite Ferreira. *Opponent Modelling in Texas Hold ' em : Learning Pre-flop Strategies in Multiplayer Tables Opponent Modelling in Texas Hold ' em : Learning Pre-flop Strategies in Multiplayer Tables.* Master thesis, Universidade do Porto, 2008.

[CHH02] Murray Campbell, A.Joseph Hoane, and Feng-hsiung Hsu. Deep blue. *Artificial intelligence*, 134(1-2):57–83, January 2002.

[Dav02] A Davidson. *Opponent modeling in poker: Learning and acting in a hostile and uncertain environment.* Master thesis, University of Alberta, 2002.

[F0́8] Dinis Félix. Artificial Intelligence Techniques in Games with Incomplete Information : Opponent Modelling in Texas Hold'em. (March), 2008.

[Hoe06] Bret Hoehn. *The effectiveness of opponent modelling in a small imperfect information game*. Master thesis, University of Alberta, 2006.

[JB09] Michael Johanson and M Bowling. Data biased robust counter strategies. *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics*, 5, 2009.

[Joh07] MB Johanson. *Robust strategies and counter-strategies: Building a champion level computer poker player*. PhD thesis, 2007.

[LIA] LIACC. LIACC.

[Liv08] Charles Livingstone. The social economy of poker machine Gambling in The Social Economy of Poker Machine Gambling in Victoria. *International Gambling Studies*, (February 2013):37–41, 2008.

[LZ06] Michael Littman and M Zinkevich. The 2006 AAAI computer poker competition. *ICGA Journal*, 2:1–2, 2006.

[McC04] Pamela McCorduck. *Machines Who Think*. A K Peters/CRC Press, 2 edition, 2004.

[Pas11] Nuno Passos. *Poker Learner : Reinforcement Learning Applied to Texas Hold ' em Poker*. Master thesis, Faculdade de Engenharia da Universidade do Porto, 2011.

[Sal] Teppo Salonen. Bluffbot.

[SB98] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, 1998.

[Sch06] TC Schauenberg. *Opponent modelling and search in poker*. Master thesis, University of Alberta, 2006.

[Skl02] David Sklansky. *Theory Of Poker - A Professional Poker Player Teaches You How to Think Like One*. Two Plus Two, 2002.

[SLLB96] Jonathan Schaeffer, Robert Lake, Paul Lu, and Martin Bryant. CHINOOK the world man-machine checkers champion. *AI Magazine*, 17(1):21–29, 1996.

[TR] Luís Filipe Teófilo and Luís Paulo Reis. Building a No Limit Texas Hold ' em Poker Agent Based on Game Logs using Supervised Learning.

[Wat70] DA Waterman. Generalization learning techniques for automating the learning of heuristics. *Artificial Intelligence*, 1(1-2):121–170, 1970.