FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

FEUP

An Intelligent Poker-Agent for Texas Hold'em

Rui André Sêca

Report of Dissertation
Master in Computing Engineering

Supervisors:
Luís Paulo Reis (professor at University of Porto's Faculty of Engineering)
Dániel László Kovács (lecturer at Budapest University of Technology and Economics)

2008, June

An Intelligent Poker-Agent for Texas Hold'em

Rui André Sêca

Report of Dissertation
Master in Computing Engineering

Approved in oral examination by the committee:

President: Armando Jorge Sousa (Professor at Faculty of Engineering in Porto's University)

External Examiner: João Balsa da Silva (Professor at Faculty of Sciences in Lisbon's University)

Supervisor: Luís Paulo Reis (Professor at Faculty of Engineering in Porto's University)

18th July, 2008

Abstract

Texas Hold’em Poker is, nowadays, the most popular poker game in casinos and online gambling websites.

Conversely to other games, like chess or backgammon, where all the information about the game state (e.g. position of all the pieces on the board) is available at any given time, in Texas Hold’em players must rely on imperfect information, for making their decisions along the way (since they do not know the hands of the others).

The complexity of the game makes its complete formal analysis (and solution) unfeasible, thus heuristics are introduced.

Expert poker players will always try to base his/her actions not only on the cards they hold, but also on an estimation of the cards they think the opponents hold. This adds an incredible amount of unpredictability into the game while the players try to adapt the best they can, and strategies such as bluffing and trapping arise. It is a hard task to create a competitive computer poker program able to play efficiently against human players.

For about a decade, research groups and hobbyists have tried different approaches to this interesting engineering problem. These approaches included knowledge-based systems, simulation, game-theoretic methods, and adaptative information game-tree search. The goal of this work was to review these approaches, and to define and implement an architecture for an intelligent agent capable of playing Texas Hold'em Poker, competitively, against human and non human agents. Its architecture extends many existing features (such as *opponent modeling*), as well as implementing new methods (e.g. a more accurate way to estimate the winning odds against multiple opponents).

The intelligent agent was then put to prove against several different agents and the results obtained are very positive, showing the agent performing well against average and strong players. Based on the assessment of results, conclusions were taken and a road map was drawn for further improving the intelligent agent.

Resumo

Poker Texas Hold'em é, hoje em dia, o jogo de poker mais popular nos casinos, reais e online.

Ao contrário de outros jogos, como xadrez ou gamão, em que toda a informação sobre o estado do jogo é conhecida (p.e. a posição de todas as peças no tabuleiro), a qualquer momento, para todos; em Texas Hold'em os jogadores têm de basear as suas decisões ao longo do jogo em informação *imperfeita* (pois as cartas dos adversários estão escondidas).

A complexidade deste jogo torna impossível a sua completa análise formal (e solução), e portanto heurísticas são introduzidas.

Um jogador experiente tentará sempre que as suas acções reflictam não só as cartas que ele possui, mas também as cartas que ele julga que os oponentes possuem. Isto faz com que este jogo se torne muito imprevísivel, à medida que cada jogador se tenta adaptar o melhor possível aos outros. Surgem assim estratégias como *bluffing* e *trapping*. Criar um programa de computador, capaz de jogar eficientemente contra jogadores humanos, é uma tarefa díficil.

Durante a última década, grupos de pesquisa e investigadores independentes tentaram diversas novas abordagens a este interessante problema de engenharia. Estas abordagens incluem sistemas de conhecimento, métodos de simulação, teoria de jogos, e pesquisa em árvore de informação adaptativa.

Esta dissertação visa cobrir estas abordagens, definir e implementar uma arquitectura de um agente inteligente, capaz de jogar poker Texas Hold'em a um nível competitivo contra agentes humanos e não humanos. Esta arquitectura implementa várias metodologias já existentes (como *opponent modeling*), assim como re-inventa algumas destas (p.e. um novo método para calcular a probabilidade de ganhar contra diversos oponentes, que oferece mais precisão).

O agente inteligente foi então posto à prova contra diversos outros agentes, e os resultados obtidos são bastante positivos, mostrando uma boa performance do agente contra jogadores intermédios e jogadores fortes. Tendo em conta os resultados, conclusões foram tiradas, e algumas perspectivas para melhorar o agente são aqui expostas.

Acknowledgements

I can think of this work as a road. Here, I want to thank those who walked this road with me.

Firstly, to my supervisor **Luís Paulo Reis**, for selecting me for this project, and showing his support right from the beginning.

To **Dániel László Kovács**, whom I met regularly to discuss the advances in my work. Our meetings were exhausting sometimes, as we both put enthusiasm and effort in our ideas. I want to show you my appreciation for your help.

To **Budapest University of Technology and Economics**, for accepting me as an Erasmus student, and promptly providing everything I needed.

To **University of Porto's Faculty of Engineering**. To all the teachers and people that helped me throughout my education.

To **Bryan Pellegrino**, professional poker player who has recently moved to Las Vegas. He has helped me occasionally with the expert knowledge of this game.

To my Erasmus friends and flat mates, who put up with me in the times of more stress, and forgave me for not going out with them when I had to work.

Finally, to my family, who have always been present to show their support and love.

Contents

List of Figures

List of Tables

Chapter 1

1. Introduction

Poker is a very interesting game from the point of view of Artificial Intelligence [39]. Its unique attributes make it different from other games, and an important (and difficult) problem to solve.

From a perspective of Game Theory [16], which was one of the first scientific approaches to it, it is a game of *imperfect information*. Unlike chess, where at each moment players have access to all the information of the game, in poker, players hold cards unknown to their opponents.

It's a game of "*luck*". Its non-determinism comes from randomly shuffling the deck at the beginning of every new game. This causes the incomplete knowledge of the players, since after the deck is shuffled and the cards dealt, each player only knows the cards he holds.

Furthermore, the outcome of a game is only *partially observed*, meaning that every time a player folds, we don't get to see his cards, and so we can't assert with certainty the strategy he employed.

For all the above reasons, poker has a high level of **unpredictability** and **deceitfulness**, as each player tries to adapt to his opponents, in order to maximize his winnings by outsmarting their opponents.

Because every action has an unpredictable outcome, actions must be chosen from a probability distribution. "*Opponent Modelling*" is the method of creating this distribution based on the information we have from the opponents.

1.1 Motivation

The motivation behind this work is that the Texas Hold'em poker-game is hitherto unsolved. Formally (e.g. from the perspective of Game Theory), no *optimal solution* has been found yet. And unlike other games like chess or backgammon, computer programs have not been able to surpass the best humans in this game. This is due to the fact that traditional methods, used in other games, are incapable of handling Poker's unique properties.

In fact, it is a very difficult engineering task to build a program capable of playing poker on the level of an average human player. One sensible aspect of the program must be to adjust itself to its opponents playing style very quickly, in an effective way. Humans make use of their intuition for this.

Regarding poker, its popularity has exploded in the beginning of the 21st century, with many online casinos making this game available to play in the Internet.

In 2005, more than US$ 60 billion was gambled on online poker casinos, with an estimate of 1.8 million players per day. This has become a gigantic industry, with effects on the high-stakes world of investing. The online poker revenues have escalated from $US 82.7 million in 2001, to $US 2.4 billion in 2005. The forecast is to hit $US 24 billion by 2010 [33].

The major motivation for this work has been to follow the most recent developments in poker AI, and to improve them in order to create an Intelligent Agent capable of good results.

1.2 Goals

The goal of this work is to create a successful intelligent poker agent. In order to do so, many steps have been taken: from researching and investigating previous successful attempts, to defining our own agent's architecture and implementing these concepts in a new way.

The implemented agent is not expected to be a winner against other well established poker agents, developed by research groups over a period of several years. However, it should be designed to use the best known methods, and try to innovate in some way. Special attention was given to the latter point, during this work.

The degree of success of such an agent is difficult to measure. Nonetheless, the assessment of results should accomplish to show us the progress in development, and an estimation of the agent's expected profit over time against other players.

The difficulty level, as well as the playing style of the agent's opponents should vary, in order to test the agent in different scenarios.

It has also been defined that the agent should be able to employ some poker strategies in his playing style, such as "*trapping*" and "*bluffing*". These strategies add uncertainty to his playing style, and make it much harder for his opponents to accurately model it. *(please refer to Appendix A on poker terms.)*

1.3 Summay of Contents

This document is divided into six chapters.

Chapter 1 is the Introduction, and gives us a big picture about the project and its context. It also provides an overview view over the rest of this document.

The second chapter is intended to introduce us to Texas Hold'em Poker, by giving us a brief explanation of how the game is played, and after discuss the problems of implementing a computer program for playing poker.

In the third chapter, previous theories and algorithms for a computer poker agent are discussed.

The most important chapter is the fourth chapter, where the developed agent is broken down to its components and described in detail. From its design to the implementation.

Chapter 5 covers the assessment of the results. The new agent is tested in different scenarios and the outcome is thoroughly discussed.

Finally, the sixth and last chapter of the thesis addresses the conclusions of this work and discusses the possibilities of future improvements.

Chapter 2

2. An Overview of Poker

Poker has been played since the 19th century [34]. Throughout time, the rules have evolved and it became more widespread. Recently, poker's popularity sky rocketed, and nowadays, poker is played in online casinos by hundreds of thousands of people per day and by celebrities on television [35].

Poker is a card game, in which players bet that the "hand" they have is better than their opponents' hand. All bets go into a pot, to be collected later by the winner.

The winner is the player who makes an un-called bet, leading all other remaining players out of the hand. If in the end of the final round, there are two or more players remaining, then the winner is the player who holds a "hand" with the highest value. (See Poker Hank Rankings chapter for details about the value of a "hand").

There are many poker variants. The most popular nowadays, and relevant to this project, is Texas Hold'em. It is the poker variant used to determine the world champion at the annual World Series of Poker.

Within Texas Hold'em, the betting structure can be No-Limit, Fixed Limit, or Pot Limit.

This work discussed in this thesis regards the Texas Hold'em variant, and is focused on the Fixed Limit structure.

2.1 Texas Hold'em Rules

This chapter is intended to cover the rules of Poker Texas Hold'em, and give an overview of this game.

Poker Texas Hold'em is a community card game, which means some cards are played face-up in the table, and can be used by all the players. Each player also holds 2 private cards, called ***hole cards***, which he/she uses together with the 5 ***community cards*** in order to make the best possible 5-card hand.

Before the game starts, one player is assigned to be the ***dealer***, and his seat is marked with a *dealer button*. This position is rotated clockwise at the end of every round.

The two players on the left of the dealer start by putting a predetermined amount of money into the *pot*. This is to ensure that there is action with every hand. It is called *posting the blinds*. The first player on the left of the dealer puts the ***small blind***, which is half the minimum bet, and the second player puts the ***big blind***, which is the minimum bet for this table.

In the image below, player "F" is the *dealer*, marked with the *dealer button*. Player "A" is posted the *small blind*, and player "B" posted the *big blind*. The seats between player "F" and player "A" are empty (awaiting players).

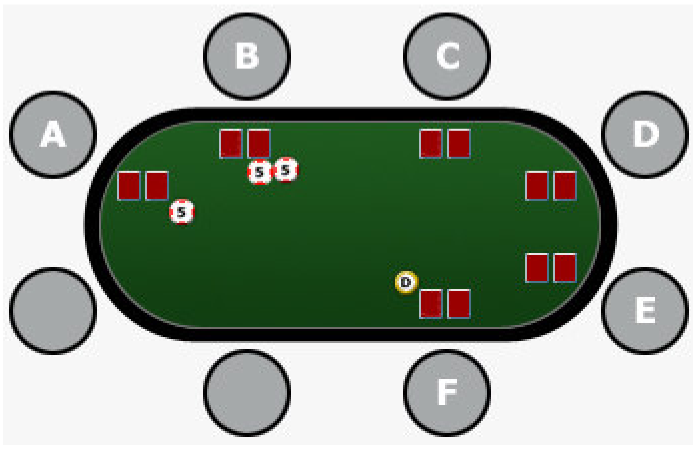


Fig 1: The dealer and the blinds.

This game consists of 4 ***betting rounds***, in which the players can act.

In every betting round, each player acts in turn, clockwise. When it is a turn of a player to act, he can decide upon several possible actions. If someone already betted money into the pot in this round, then the player can:

- **Call** – Match/equal the other player's bet.
- **Raise –** To increase another player's bet.
- **Fold –** Forfeit cards and thus giving up on the pot.

If no one betted so far in this round, then the player's options are the following:

- **Check** – Passing on making an action.
- **Bet –** Put money in the pot, and so force the opponents to *Call* the bet if they want to continue in this hand.
- **Fold –** Forfeit cards and thus giving up on the pot.

The players continue to act, in each round, until everyone matches the amount betted/raised by a player. The player that bets may only act again in the same round if another player *re-raises* him.

The first betting round is called ***Pre-Flop***, and the players are given 2 cards faced down, which remain private for each player's opponents. These are the ***hole cards*** (also called ***pocket cards***).

And so the betting round starts, and the first player to act is on the left of *big blind* seat. This player has to decide whether to *Call* the *big blind*, *Raise*, or simply *Fold* his cards.

Logically, the only player who is given an opportunity to *Check* in the *Pre-Flop* phase is the *Dealer*, if nobody else raised.

After this betting round is complete, come 3 more betting rounds, called the ***Post-Flop*** rounds. From now on, the first player to act is always the player on the left of the dealer.

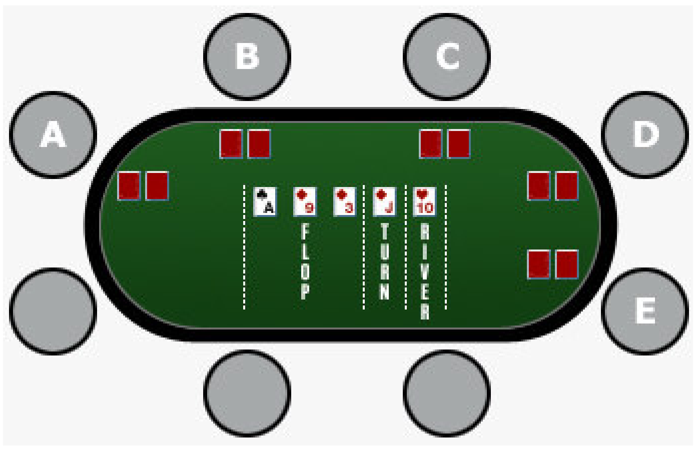


Fig 2: Post-Flop betting rounds.

The second *betting round* is called ***Flop***. Three community cards are dealt face up to the middle of the table. These cards are shared by all the players and can be used in combination with *hole cards* each player holds.

So the second round of bidding begins, and only ends when all the players put the same amount on the pot, or fold.

When this round is finished, the dealer turns 1 more community card face up in the table, and a new betting round starts. This third round is called the ***Turn***, and proceeds similarly to the previous round.

The fourth round is the ***River***. The last community card is dealt and displayed on the table, starting a new betting round.

In all the previous rounds, if all but one player folds, then the remaining player is declared the winner of this hand and wins the *pot* on the table. At this point, this player may choose to show his hand to the players, or ***muck*** it, which means to throw the hand away without showing anyone what it was.

If after the *River*, there are still two or more players in the hand, a ***Showdown*** phase takes place. In this phase, all players show their cards, starting with the last person to bet. However, after the first player shows his cards, other players may choose to *muck* their hand, which is basically the same as folding. This is an important part of poker as you can *muck* to keep other players from learning your playing style.

Players can use any combination of seven cards – the five community cards and their two hole cards – to form the best possible five-card poker hand.

The player with the best hand wins the pot. *(See chapter below).*

2.1.1 Poker Positions

One important aspect of poker to retain regards the position of the players in the table. The position in which you play a hand, relative to the ***dealer*** (or ***button***) can be as important as the cards you hold.

This is due to the fact that the later you are to act in a hand, the more players have acted before you, and so you have gathered information about their actions.

Besides defining some seats as ***Button***, ***Small Blind*** and ***Big Blind***, we can classify the players' seats in a hand in 3 categories: ***Early Position***, ***Middle Position*** and ***Late Position***.

Let's consider a poker table with 10 players. This is called a ***full-ring*** table.

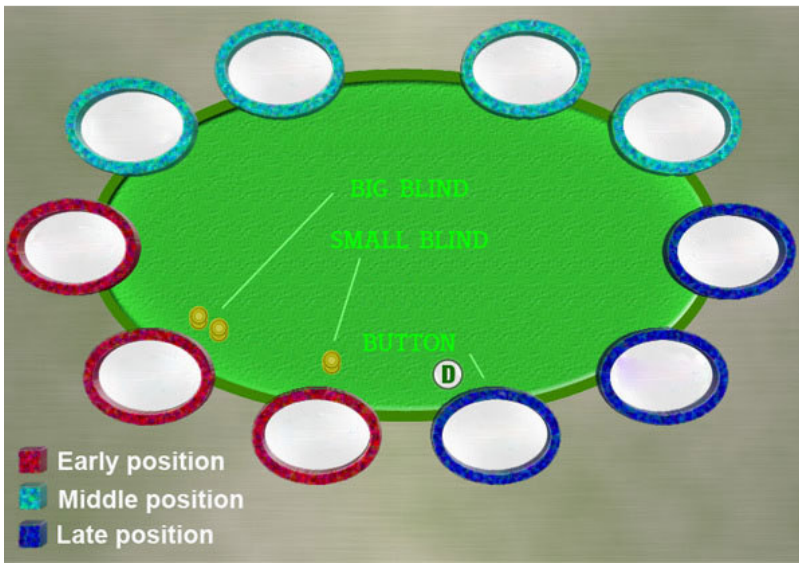


Fig 3: Positions of the players at a poker table.

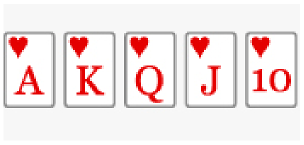
In this scenario, the first three players (after the Button) would be in Early Position. The four next players would be said to be in Middle Position, and finally the last 3 remaining players would occupy a Late Position.

The exact number of players in each of these categories changes according to the number of players in the table, but the essence of this classification is still valid.

Note: Besides ***full-ring*** tables, poker tables can be characterized as ***heads-up*** table (2 players) and ***short-handed*** table (3-6 players).

2.2 Poker Hand Rankings

In order to determine the winning hand, we have to able to rank hands. Here is a list of five-card poker hands rankings in descending order of strength.

	**Royal Flush** This is the best possible hand in standard five-card Poker. Ace, King, Queen, Jack and 10, all of the same suit.
	Straight Flush Any five-card sequence in the same suit.

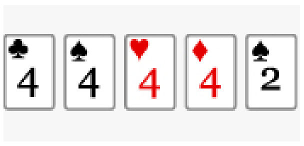
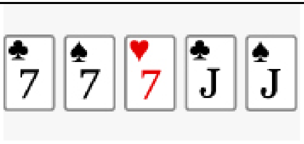
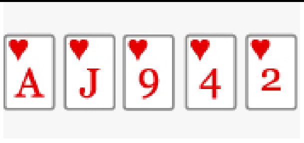
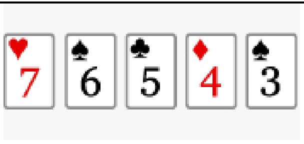
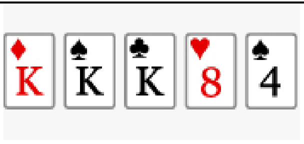
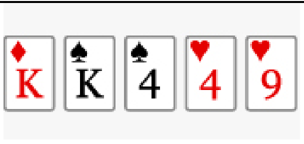
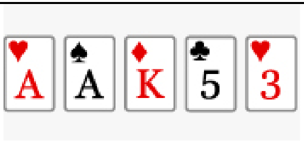
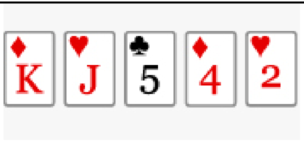
	**Four of a kind** All four cards of the same value (or rank).
	**Full house** Three of a kind combined with a pair. Ranked by the *trips* (three of a kind).
	**Flush** Any five cards of the same suit, but not in sequence. Ranked by the top card.
	**Straight** Five cards in sequence, but not in the same suit. The ace plays either high or low in the straight. Ranked by the top card.
	**Three of a kind** Three cards of the same value.
	**Two pair** Two separate pairs, and one *kicker* of different value. The *kicker* is used to decide upon a tie of the same two pairs.
	**One pair** Two cards of the same value. Three *kickers*.
	**High card** Any hand that does not qualify as one of the better hands above. Ranked by the top card, then the second card and so on.

Table 1: Five-card hand ranks.

Note: In Texas Hold'em, suits are not taken in consideration when ties occur with the best five-card hand. In the case of a tie, the pot is split equally among the winning hands.

2.3 Fixed Limit Variant

Fixed Limit is one of the possible variants within Poker Texas Hold'em.

Contrarily to the No-Limit variant, the bet amount is fixed, and so the players can only choose the action to take, having no decision on the amount to bet.

Poker tables have two pre-defined values: ***small bet*** and ***big bet***. The *small bet* is the same as the ***minimum bet*** (mentioned above). A poker table would typically show these values in the format "**$1/$2**", meaning, in this case, that the small bet is $1 and the big bet is 2$.

In Fixed Limit, in the two first betting rounds (**Pre-Flop** and **Flop**), the bet value is defined to be the small bet, and in the remaining two rounds (**Turn** and **River**), it is equal to the big bet.

Another rule of this variant, is that the maximum number of raises is limited to three. To demonstrate this, let's consider the following example: a poker table "**$1/$2**", in the **Turn** stage, with only 3 remaining players in the current hand. The action goes as follows:

- **Player "A"** bets $2. (lays down $2 in the table)
- **Player "B"** raises $2. (lays down $4 in the table, to include Player "A" bet)
- **Player "C"** re-raises $2. (lays down $6 in the table, to include former bets)
- **Player "A"** has now two options if he wants to continue playing this hand: either **Calls** the bets that Player "B" and Player "C" made, to a total of $4, or re-raises $2 (putting this way more $6 in the table).

If Player "A" decides to re-raise, then he is said to ***cap*** the pot. This means that the remaining players in the hand will only be able to **Call** Player's "A" re-raise (if they wish to continue in this hand,) but they won't be allowed to re-raise anymore in this round. And so after Player "A" re-raise, both Player "B" and Player "C" would have to pay more $4 (two big bets) to remain eligible to win the this hand's pot.

2.4 Computer Poker

As already referred, the game of poker has interesting, unique properties that can't be found in other games. Due to this fact, poker has been used as a testbed in different areas [8], and provides excellent challenges to decision making under conditions of imperfect information [1].

Creating a computer program capable of playing poker in a world-class level poses itself as a challenge. In fact, so far, the poker programs haven't evolved to the point of surpassing the best human poker players, in any variation of Texas Hold'em [32]. This is due to a number of reasons.

The scientific community has been approaching this problem in different ways, over a number of years. From those contributions, one must be emphasized: the *University of Alberta Computer Poker Research Group* (*CPRG*) with numerous papers and thesis on the poker game-playing AI [11]. These approaches, as well as the *CPRG* research results, methodologies and algorithms, will be addressed in the next Chapter.

From a game-theoretic perspective, it is unfeasible to compute an *optimal solution* to the problems in the poker's domain. In fact, even the two-player Texas Hold'em game has a search space size of $O(10^{18})$ [15].

The outcome of every action in poker depends immensely on the actions of the other players and the accuracy of the model we constructed of them. Even an ***optimal*** poker strategy (a strategy that minimizes loss against *any* possible opponent's strategy) could prove to be not as effective or profitable as another one (***maximal*** strategy), which would better exploit the opponents' weaknesses [1, 15]. This is the reason why **Opponent Modelling** is a main component of any good poker program.

Furthermore, humans make use of their intuition in poker, and so, change their playing style very fast to adapt to their opponents style and to avoid being accurately modelled. This issue was discussed in the past, as "*tracking a moving target*" [13].

On the other hand, computers are able to make use of their processing speed and calculate statistics and probabilities that help them to make decisions. While this is true, the possibilities in a poker hand grow exponentially, and the use of processor speed is limited when faced with expensive calculations, as we'll see in the next chapter.

These computations are also limited by the time to act. When playing poker online, each player has a limited time to act, and in order to make the game fluid, poker AI agents' aim at a response time of a few seconds per action, at most [1]. This is also important when assessing results, as the chance element plays an important role in the outcome of a hand, numerous trials must be made in order to observe useful properties.

Moreover, the task of assessing a poker agent's performance is not a trivial one. It should be tested in a large diversity of scenarios. These scenarios should include human players with different playing styles. However, establishing dozens or hundreds of thousands trials with human players can be very difficult and time-consuming.

So far, the best poker agents have focused on a very specific variant of Texas Hold'em, in order to attain a good result. Previous agents have focused mostly on Limit Heads-up tables, Limit Full ring tables, and more recently No-Limit Heads-up tables.

In conclusion, Poker Texas Hold'em has great complexity, and tackling effectively the problem of building a high performance computer program has proved to be a hard challenge in the domain of Artificial Intelligence, yet to be solved.

Chapter 3

3. Previous Approaches

As mentioned before, in the last decade computer poker has become a center of attention to the scientific community.

There have been many approaches and developments in this field, and this chapter is meant to give an overview of the former approaches and methodologies, up to the state of the art.

Most of the literature contributions have come from the *University of Alberta Computer Poker Research Group* (*CPRG*) [11].

3.1 Deterministic Rule-based methods

One intuitive approach is to design a set of rules that handle specific situations in the game play.

This type of "expert system" has been proven efficient in games where there are only a few distinct cases to be handled. However, poker has a very rich context, and trillions of distinct scenarios may arise during a game [4].

Furthermore, this system is also limited by the knowledge of the domain expert, for it can only be as strong as its creator.

These limitations have been historically proven, by the program *Turbo Texas Hold'em*, written by Bob Wilson [4]. After more than 15 years of development, this computer program was able to handle thousands of different scenarios. Even so, it can be easily defeated by average players.

A *deterministic rule-based approach* will always decide upon the same action, given a specific context of the game. In game theory this is known as a *pure-strategy*. Due to this fact, opponents can easily model the gameplay of an agent that implements this approach, and rapidly exploit it.

Unpredictability is one important aspect in poker, and strong players must be able to change their playing style over time.

For example, if we know that our opponent always *raises pocket Aces* in the pre-flop stage, we can correctly infer that he doesn't have this hand if he takes another action in this stage of the game. These inferences about the opponent's hand are useful to correctly decide upon an action, and therefore become profitable.

A more general method is the *probabilistic rule-based approach*, which defines a randomized *mixed strategy* for a specific context (e.g. a probability distribution over a set of possible actions). This approach is able to handle unpredictability much better.

3.2 Probabilistic Formula-based methods

This approach is much less rigid than the *deterministic rule-based approach*, for it abstracts different scenarios into a much smaller number of circumstances. This abstraction is made by using formulas, which consider several variables pertaining to strategic elements in the game.

However, this approach still runs on the same limitations as the rule-based approach, even if to a lesser extent.

The most robust and well-known agents that implement this approach are Loki and Poki, both developed by the CPRG.

The following sub-chapter will go over the main aspects of these agents.

3.2.1 Loki and Poki's architecture

Loki, and its successor Poki, are agents designed to play in the variant Limit Texas Hold'em, and play best in a full ring table. They were developed by the CPRG.

These agents introduced some new, important, methodologies such as:

- Pre-Flop Hand Evaluation.
- Post-Flop Hand Ranking.
- Hand Strength, Hand Potential, Effective Hand Strength.
- Opponent Modelling.

The betting strategy of these agents is divided between before the *flop* and after the *flop*, and is significantly different. Before the *flop*, the agent's decision is much simpler, since it only considers its private, two hole cards, as opposed to the *post-flop* decision, whereas it has to analyse how its own cards combine with the community cards already revealed on the table.

Pre-Flop

In the *pre-flop*, there are $\binom{52}{2} = 1326$ different hands.

However, since many of these are equivalent before the flop (e.g. 3♢6♢ is equivalent to 3♣6♣), we can narrow down to 169 distinct hand types (13 paired hands, 78 suited hands and 78 unsuited hands).

The value of each of these hands is called an income rate, and is computed based on a technique known as ***roll-out simulation***. This method is done offline, and consists of playing several million games (trials) where all players call the first bet. All hands then proceed to showdown without any further betting.

This method is also commonly referred to as *all-in equity*, since after calling the first *bet*, all players are assumed to be *all-in*.

This method provides an approximation of the expected value of the *pocket hands*, and is useful to decide upon a strategy in the pre-flop phase.

A refinement to this technique is the ***iterated roll-out simulation***. It is done by repeating iterations of this technique, but deciding the action for each player in accordance with the result of the previous iteration. This means that the weakest hands will be folded *pre-flop*, providing a better estimation of the real expectancy of each hand. This is based on the assumption that most weak hands never get to see the flop.

The results obtained are quite similar to the group rankings suggested by professional player David Sklanksy, and are shown in the table below.

Group 1		**Group 2**		**Group 3**		**Group 4**	
+2112	AA	+714	TT	+553	99	+481	T9s
+1615	KK	+915	AQs	+657	JTs	+515	KQo
+1224	QQ	+813	AJs	+720	QJs	+450	88
+935	JJ	+858	KQs	+767	KJs	+655	QTs
+1071	AKs	+718	AKo	+736	ATs	+338	98s
				+555	AQo	+449	J9s
						+430	AJo
						+694	KTS
Group 5		**Group 6**		**Group 7**		**Group 8**	
+364	77	+304	66	+214	44	-75	87o
+270	87s	+335	ATo	+92	J9o	+87	53s
+452	Q9s	+238	55	+41	43s	+119	A9o
+353	T8s	+185	86s	+141	75s	+65	Q9o
+391	Kj0	+306	KTo	+127	T9o	-129	76o
+359	QJo	+287	QTo	+199	33	-42	42s
+305	JTo	+167	54s	-15	98o	-83	32s
+222	76s	+485	K9s	+106	64s	+144	96s
+245	97s	+327	J8s	+196	22	+85	85s
+538	A9s			+356	K8s	-51	J8o
+469	A8s			+309	K7s	+206	J7s
+427	A7s			+278	K6s	-158	65o
+386	A6s			+245	K5s	-181	54o
+448	A5s			+227	K4s	+41	74s
+422	A4s			+211	K3s	+85	K9o
+392	A3s			+192	K2s	-10	T8o
+356	A2s			+317	Q8s		
+191	65s						

Table 2: Income Rate Values vs. Sklansky Groups.

There is a strong correlation between Sklansky groups and the results of the *roll-out simulations* (table 2). CPRG preferred to use the computed results [4].

The abstraction of these hands to single income rates comes with some limitations. Some hands with lower expectancy can be played under special circumstances (e.g. for one bet to call in a late position, after many players called). Also, there are certain hands that can easily draw into a very strong hand, such as a flush or a straight. These are called *drawing hands.*

So, after Loki/Poki has received its *hole cards*, it looks up the table for the *income rate* value of the hand.

Then, it uses a formula-based approach to govern the betting strategy for the *pre-flop*. This system makes use of expert knowledge that has been defined by a poker expert.

Post-Flop

For the *post-flop* stages, Loki/Poki computes several variables such as *hand strength* (**HS**), *positive potential* (**PPot**), *negative potential* (**NPot**), and *effective hand strength* (**EHS**). These variables are used to assess Loki/Poki's (hereafter only referred to as Poki) hand value relative to the board (*community cards*).

Hand strength is an estimation of the probability that our hand is better than the hand of a given opponent. A simple calculation of the HS is to enumerate all other possible hands and compare them to ours, using a *Hand Ranking* function.

After the flop, there are only more $\binom{47}{2} = 1081$ distinct hands an opponent might hold. By computing the number of times our hand wins, loses and ties against every possible opponent's holdings, we get the probability of currently having the best hand, against a random hand.

Figure 4 shows the algorithm for a simple HS raw calculation.

```
HandStrength(ourcards,boardcards)
{
        ahead = tied = behind = 0
        ourrank = Rank(ourcards,boardcards)
        /* Consider all two-card combinations of the remaining cards. */
        for each case(oppcards)
        {
         opprank = Rank(oppcards,boardcards)
         if(ourrank>opprank)        ahead += 1
         else if(ourrank==opprank)   tied += 1
         else                       behind += 1
         }
        handstrength = (ahead+tied/2) / (ahead+tied+behind)
        return(handstrength)
}
```

Fig 4: Hand Strength calculation.

This value is un-weighted, since we're considering that the opponent is equally likely to hold any of these 1081 possible hands. However, this enumeration can take into account our belief about the possible hand of the opponent. Instead of adding one to each counter for *ahead*, *tied* and *behind*, we can add a weight representing the probability of the opponent holding a specific hand.

Weighting the enumerations will be further discussed below.

The HS estimation is an important measure, but only reflects the current situation. Since in the *flop* there are still two more board cards to be revealed, and in the *turn* one more, the ***hand potential*** calculation is introduced.

Hand potential is divided in positive potential and negative potential.

PPot is the chance that a hand, which is currently not the best, will improve to win at the showdown. **NPot** is the chance that a currently leading hand ends up losing.

Similarly to the HS calculation, PPot and NPot are calculated by enumerating all possible opponent's hands and all future board cards to come, and count how many times the hand is behind, but ends up ahead (PPot), and the number of times the hand is ahead but ends up behind (NPot). The algorithm is given in Figure 5.

```
HandPotential(ourcards,boardcards)
{
/* Hand Potential array, each index represents ahead, tied, and behind. */
       integer array HP[3][3] /* initialize to 0 */
       integer array HPTotal[3] /* initialize to 0 */

       ourrank = Rank(ourcards,boardcards)
/* Consider all two-card combinations of the remaining cards for opponent. */
       for each case(oppcards)
       {
              opprank = Rank(oppcards,boardcards)
              if(ourrank>opprank)        index = ahead
              else if(ourrank=opprank)   index = tied
              else                       index = behind
              HPTotal[index] += 1

              /* All possible board cards to come. */
              for each case(turn)
              {
                     for each case(river)
                     { /* Final 5-card board */
                     board = [boardcards,turn,river]
                     ourbest = Rank(ourcards,board)
                     oppbest = Rank(oppcards,board)
                     if(ourbest>oppbest)         HP[index][ahead] += 1
                     else if(ourbest==oppbest)    HP[index][tied] += 1
                     else                        HP[index][behind] += 1
                     }
              }
       }
/* PPot: were behind but moved ahead. */
PPot = (HP[behind][ahead] + HP[behind][tied]/2
+ HP[tied][ahead]/2) / (HPTotal[behind]+HPTotal[tied]/2)
/* NPot: were ahead but fell behind. */
NPot = (HP[ahead][behind] + HP[tied][behind]/2
+ HP[ahead][tied]/2) / (HPTotal[ahead]+HPTotal[tied]/2)
return(PPot,NPot)
}
```

Fig 5: Hand Potential calculation.

Computing hand potential may be crucial, but is also expensive, given the real-time constraints of the game (about one second per decision). In the flop, if we enumerate all possible scenarios to come (990 possible cards for *turn* and *river*), may prove to be as slow as 2000 milliseconds per computation [1].

In practice, a fast approximation of the PPot calculation can be used, such as only considering the next card to come (one card look-ahead). This decreases the calculation time to about 280 milliseconds [1].

Also, in hand potential calculation, we can weight the enumeration of the possible opponent's holding, with a probability (discussed below).

The **effective hand strength** (EHS) combines hand strength and potential to give a single measure of the relative hand strength against an active opponent.

CPRG suggests that the NPot is not as important as PPot for betting purposes, and so only uses the first for in the EHS calculation:

$$EHS = HS + (1 - HS) \times PPot \tag{1}$$

By not accounting for the NPot, Poki bets his hand more aggressively, despite good draws being possible for his opponent. As a consequence the opponent will either fold or pay to draw.

These calculations are with respect to one opponent, but can be extrapolated to multiple opponents by raising it to the power of the number of active opponents. Using a raw (un-weighted) EHS, we can account for n active opponents, by generalizing the last equation to:

$$EHS = HS_n + (1 - HS_n) \times PPot \tag{2}$$

In the case we are considering the opponent's playing style, and thus a different probability distribution over possible hands of the opponent, then this formula can be generalized to be made in respect to each opponent *i*:

$$EHS_i = HS_i + (1 - HS_i) \times PPot_i \tag{3}$$

The EHS value is a very important factor when deciding on an action to take, as it estimates our winning probability.

Weighting the Enumerations

The calculations of Hand Strength and Hand Potential assume that the opponent is equally likely to hold any possible two card combination. This is an incorrect assumption.

For example, after the pre-flop stage, the probability of an opponent holding high ranked cards is bigger than holding a hand like 7♣2♡.

Poki handles this by maintaining *weight tables* for every active opponent. A weight table enumerates all possible hands an opponent might hold and assigns to each of them a probability. This probability reflects Poki's belief that the opponent holds a specific hand.

Every time an opponent takes an action, the weight table is updated to reflect this action, by using Opponent Modeling. Opponent Modeling will be discussed in the next chapter.

Poki uses a weighted Hand Strength and Hand Potential calculation, by considering the weight tables of each opponent in the computation.

Post-flop betting strategy

After the former calculations, Poki's decision is managed by a set of betting rules, defined by a poker expert.

These betting rules take into account factors like ***pot odds***, ***implied odds***, relative betting position, betting history of the current game, etc [4].

In Loki-1, the output of this rule-based or formula-based betting strategy would dictate the decision of the agent for the situation. However, since Loki-2, the new system makes use of *probability triples*. The formula-based betting strategy outputs a *probability triple*, with the probability for fold, check/call, and bet/raise. Poki then generates a random number in the range of zero to one, and uses it to choose an action according to the *probability triple*. This makes Loki-2 and Poki much more unpredictable, as it can opt for a different strategy in the exact same context.

The details of the expert knowledge in the betting strategy of Poki are unknown, as CPRG opted not to discuss these [4].

3.2.2 Opponent Modeling

Opponent Modeling is a crucial component of a good poker program.

While in games such as chess, this particular aspect is not necessary to achieve a world-class level of play, in poker, opponent modeling directly addresses the information about the game, which is unknown to the players (opponent's hand, cards to come, etc…), and the deceitfulness of the opponents (strategies like bluffing and trapping).

One must be able to predict opponents' play, and also to avoid being predicted.

Opponent Modeling is used by agents such as Poki (discussed previously throughout Section 3.2.1).

Poki uses it in two different ways: to deduce the strength of an opponent's hand, based on his betting actions, and to predict their action in a given situation of the game.

Opponent modeling can use a fixed strategy, and therefore be independent of the opponent in question – *Generic Opponent Modeling*, or can be relevant to each specific opponent – *Specific Opponent Modeling*.

Statistics-based Opponent Modeling

One intuitive way of predicting a player's actions is to expect him to behave the same way as he did in the past. For example, if a player is known to bet 40% of the time immediately after the flop, then one could infer that the player normally bets with the top 40% of their hands, in the same exact situation.

The first opponent modeling technique implemented by Poki was a statistical table, which collects the betting frequencies of the opponent, in a variety of contexts.

The context is defined by some important factors, such as the betting round (pre-flop, flop, turn, river), bets to call (zero, one, two or more), etc…

Many actions must be observed for the prediction to converge to a useful value. So, a context can neither be defined too narrow, since it will take too long to collect enough samples for each scenario, or too broadly, as it will fail to capture relevant information from the different circumstances.

Another concern of defining a context too narrow (defined by many context variables), is that while it's still in the process of gathering observations from the opponent's play, the opponent might already be changing his strategy, rendering the prediction useless.

Neural Networks-based Opponent Modeling

Artificial Neural Networks (ANN) have been known for their ability to learn and identify patterns in noisy data [13]. In the past, they have been applied to computer poker, with the goal to create a more general system for opponent modeling.

In 1999, an ANN that consisted of nineteen input variables (context items such as the number of players, game stage, etc…), and three outputs (fold, check/call, bet/raise) was trained offline with data from human players. [13]

The objective of this experiment was to identify factors that either influence, or are correlated with a player's next action.

Two particular strong features for prediction were identified and added to the existing statistical opponent modeling, to create new contexts. This improved version proved to be better than the former one [6].

The efficiency of the neural network was also put to trial, and showed promising results.

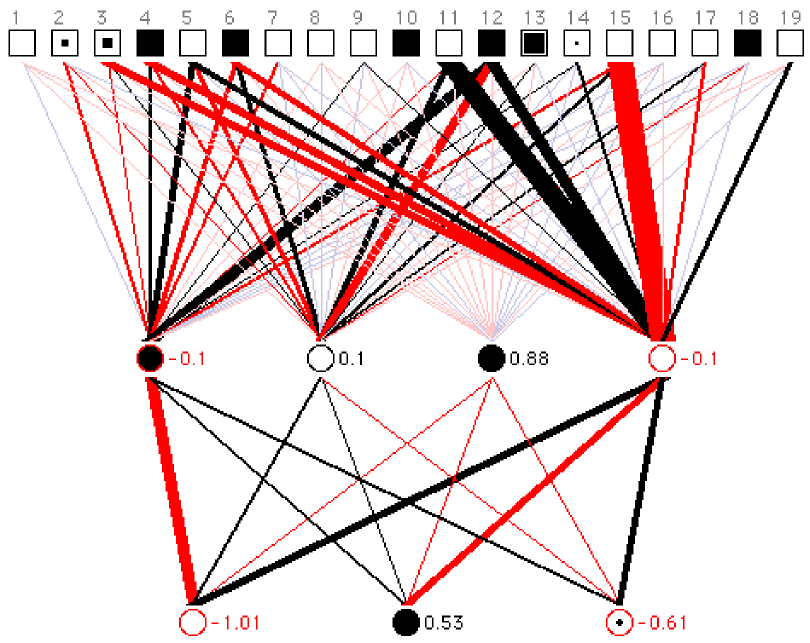


Fig 6: A Network after being trained. (Shown correctly predicting a call)

ANNs have proved themselves to be a useful way to model poker players, with a predicting accuracy higher than in the previous opponent modeling systems. However, neural networks have only been used as an off-line technique, and may not be feasible in real-time. CPRG has been experimenting using a real-time neural network system to replace the frequency table entirely [4].

3.3 Simulation methods

A poker program based on expert rules is limited by the prohibitively large domain space of poker. In fact, having rules covering every relevant situation in detail is not feasible.

An alternative to having a betting strategy that relies on expert knowledge, is to have a *simulation-based* betting strategy.

Simulation is defined as the repetition of many *trials*, in order to obtain a statistical average. In a Monte-Carlo simulation each trial consists of randomly selecting from the complete domain of possibilities.

One example of a simulation is the *roll-out simulation* discussed in the preflop betting strategy of Poki, to estimate income rates of the starting hands.

Poki supports a *simulation-based* betting strategy, since the early version of Loki-2. When faced with a decision, Poki invokes a simulation routine that assigns different hands to his opponents and plays out the game from the current state until the end, *n* number of trials. This routine helps Poki decide which action to take (Fold, Check/Call, or Bet/Raise), by selecting the action with the highest expected value.

The expected value of a Fold decision can be calculated without simulation since there is no future profit or loss. For the other two decisions, this routine estimates their expected values. In order to do so, each trial is played out twice: the first one considers the consequence of a check/call, and the second one considers that Poki bets/raises. These expected values are computed by calculating the amount of money Poki wins or loses by the end of the trials.

Throughout the simulation, the future actions of the opponents can be selected randomly from the possible actions or selected from a probability distribution, after consulting the Opponent Modeling component.

After many trials the expected value of these decisions will start to converge. However, with randomly sampling the opponent's possible hands, it can take a long time for the simulation to converge on accurate estimates.

To cover for this, Poki implements *Selective Sampling,* which consists on selecting hands for the opponents based on a probability distribution. This probability distribution reflects the likelihood of an opponent holding each possible hand, and is computed by using *weight tables* maintained for each opponent.

Selective sampling and simulation-based betting strategy has been experimentally proven to get better results than a simple formula-based betting strategy [8], and to inherently deal with complex strategies such as *check-raising* without providing any additional expert knowledge [4].

3.4 Game-Theoretic methods

In game theory, all two-player zero-sum games have at least one *equilibrium strategy*. Playing an *equilibrium strategy* ensures that an agent will obtain at least the game-theoretic value of the game, and that the other players can not benefit by changing their own strategy unilaterally. In the long run, an agent that implements this strategy will not lose, and might profit due to opponent's errors.

John Nash extended the idea of *equilibrium strategies* to N-person games, using poker as an example [38]. Unfortunately, finding exact *equilibrium solutions* is limited to relatively small problem sizes, and is not practical for most real domains.

Some abstraction techniques have been used in the past to reduce a two-player Texas Hold'em game space of 10^{18} to a highly abstracted model with a game space of 10^{7}. This model captures the most essential properties of the real domain, such that an exact *equilibrium strategy* can be computed and mapped back onto real poker. This has proved to be a very useful approximation of an *equilibrium strategy* for the real domain [14].

These solutions were used to create substantially improved poker-playing programs by the CPRG. This class of programs, collectively called PsOpti or Sparbot, was able to defeat strong human players and be competitive against world-class opponents [14].

Indeed, Sparbot proved to be hard to intimidate, and showed good results in his game against world-class player Gautam Rao [4]. However, this agent also showed some flaws, as this approach only gives a crude approximation of a true equilibrium strategy.

It is believed that over time an experienced poker player can discover subtle tendencies in Sparbot's playing style, and efficiently exploit them [4].

Nevertheless, this approach represented a large leap forward in the abilities of poker-playing programs.

3.5 Adaptative Imperfect Information Game-Tree Search

As discussed in the last chapter, if there was a program based on an exact *equilibrium solution*, then no human or computer player could expect to defeat it, in the long run. However, finding an exact solution for a game with the dimensions of poker will be unfeasible in the foreseeable future [15].

Furthermore, as mentioned before, a poker-program that would implement this strategy would only ensure that it could not be defeated. In order to convincingly win against other players, one must exploit their mistakes, and rapidly adopt counter-strategies.

This is the difference between *optimal* play and *maximal* play. An agent that implements a *maximal* strategy would try to exploit their opponent's weaknesses in order to maximize profit.

Agents that implement the Game-Theoretic approach have a practical limitation by using a *fixed strategy*.

Adaptative Imperfect Information Game-Tree Search approach deviates itself from searching an optimal strategy, to attempt to exploit perceived patterns or biases in the opponent's playing style and by adapting to dynamically changing conditions. In order to do so, it uses built-in data structures for Opponent Modeling.

Two algorithms were implemented: *Miximax* and *Miximix*. These algorithms compute the expected value (EV) at decision nodes of an imperfect information game tree. And thus, are able to handle efficiently the stochastic element of poker.

The difference between the two algorithms is that Miximax always chooses the action with the highest EV. This is called a *pure strategy*, and leads to predictable play that can be exploited by an observant opponent. On the other hand, *Miximix* uses a *mixed strategy* by selecting the action from a probability distribution.

This architecture has been implemented in the computer program Vexbot. This agent defeats Sparbot and PsOpti convincingly and poses a much tougher challenge for strong human players [4].

It learns to defeat all known programs by a large margin, over time. However, its limitation also comes from this, as its performance is much worse in the beginning of a match, until enough data has been collected [15].

Chapter 4

4. A new approach: HuBot

As stated in the Introduction, a new agent was built during the course of this work. This agent that goes by the name of HuBot implements a *probabilistic formula-based* approach with opponent modeling.

Despite being based on the award winner poker agent Poki, it diverges itself from any other agent by using new methods that have been later proved to be efficient, either empirically (by observing the decisions of the agent) or experimentally (by analysing the winning rate after thousands of games).

HuBot is an agent capable of playing Texas Hold'em in the **Limit** variant. It has been designed to play best in a full ring table. This was decided because (a) there are other agents capable of playing world-class in a table with few players, and because (b) in a full ring of players some multi-player complexities arise, that the algorithms must consider and which have not yet been addressed properly.

Being Poki the state of the art full ring Limit Texas Hold'em computer agent, developed by a team over many years, it was only natural first to arm HuBot with the same successful techniques, in order to surpass them.

As a good poker program must be able to make good decision in a very short time period, some new methods proposed in this research are meant to improve the performance of the agent, while others are intended to improve the efficiency.

4.1 HuBot's Architecture

The architecture of HuBot consists of several components that interact with each other, creating an information flow consistent with the underlying framework.

An overview of HuBot's architecture is shown in Figure 7.

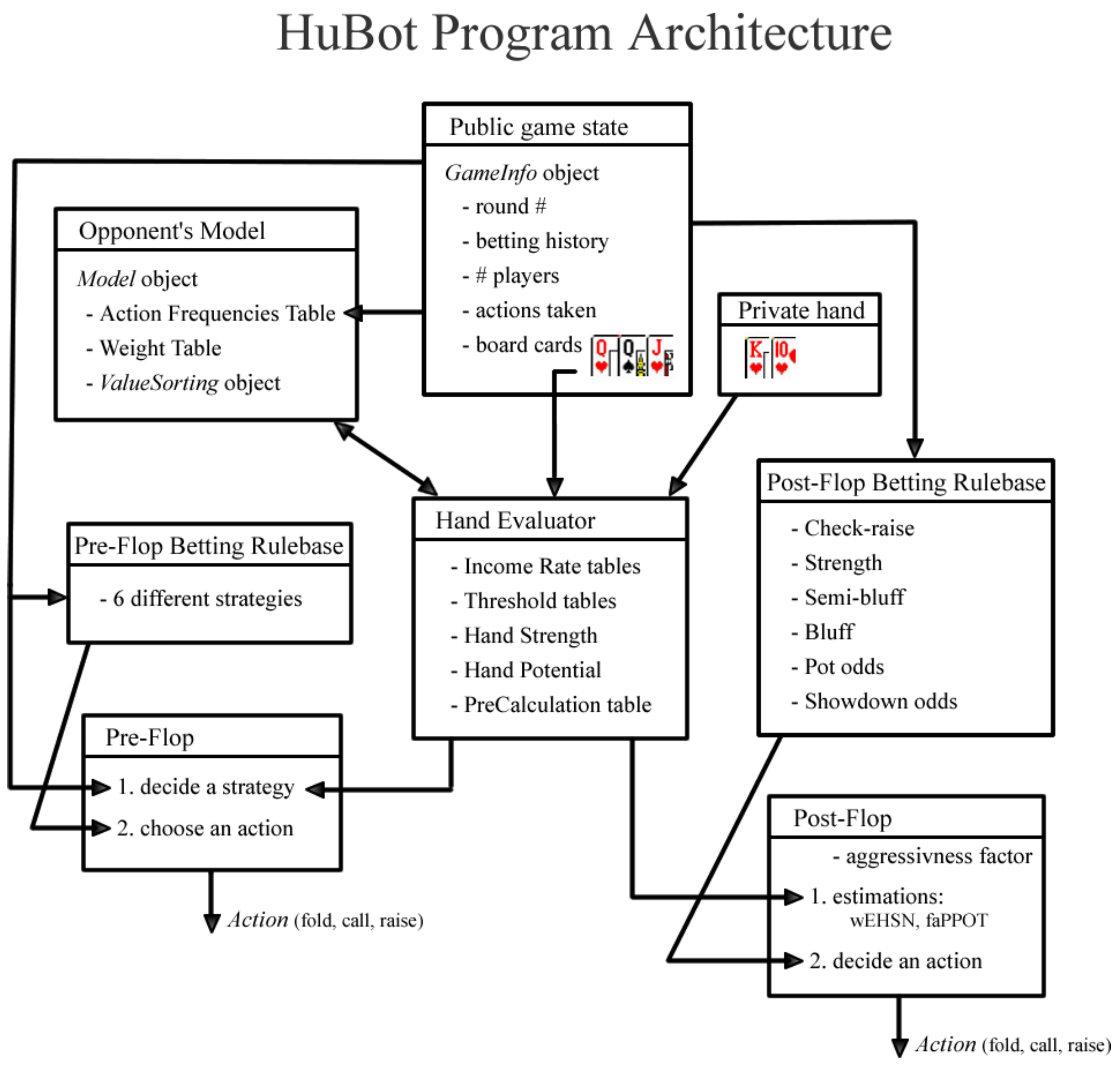


Fig 7: The architectural concepts of HuBot.

In general, the program's components are related to either the betting strategy in the pre-flop, or the betting strategy in the post-flop phase of the game.

Before the flop, there isn't much information to account for, since there aren't any community cards yet. So, more attention is given to the post-flop betting strategy, as a relatively simple expert system is capable of playing well in the pre-flop phase [4].

Firstly, the agent's private hand is assessed, by using the "Income Rate tables" and the "Threshold tables". One strategy is selected, from a fixed set of strategies, based on the value computed for the hand and the position of the active players. The strategy is then used to select an action in the pre-flop, when HuBot is presented with a decision.

This pre-flop procedure is explained in detail in Section 4.3.

In the next stages of the game (Flop, Turn and River), each time is HuBot's turn to act, its decision follows the same, single path. Section 4.4 describes it in detail.

Initially, some important values are estimated, representing the Hand Strength and Hand Potential of the agent's hand against his opponents. After, these values go into a more complex formula-based system, and an action is decided. The calculation of these values is biased, taking into account the probability distribution of the hands the opponent might hold. This probability distribution is maintained for each player in a *weight table*, and updated after an action has been observed during game play.

Furthermore, the opponent model of each player also includes a table with the action frequencies of the player, used in the re-weighting system. This includes HuBot itself.

Opponent modeling and the re-weighting method are topics of discussion in Section 4.5.

4.2 Meerkat API

The agent HuBot has been developed in Java, under the Meerkat API framework, from Poker Academy Pro 2.5 [37]. This API allows programmers to plug in their own custom agents, providing a stable framework that handles all game state information and more.

The public game state information and history is kept in the object ***GameState***, which can be queried by accessing any of its methods. Also within this object, one *PlayerInfo* object is stored for each player, in order to access public information about a specific player.

Other important data structures defined by the Meerkat API are the classes *Card*, *Hand* and *Deck*. A *Card* is defined by a rank {0 to 12} and suit {0 to 3}, therefore abstracting all 52 cards of a normal deck. A *Hand* stores several *Card*'s and provides methods to handle these.

In order to use this framework, all agents must implement the provided interface *Player*. The *Player* objects (representing both human players and computer players) are each given cards in the beginning of the game, and prompted for decisions, when necessary. These decisions come in the form of the object *Action*.

Some important events are fired when it requires HuBot's attention. These are:

- `public void holeCards(Card c1, Card c2, int seat) {`
  An event called to tell us our hole cards and seat number.

- `public Action` **`getAction`**`() {`
  Requests an *Action* from the player. Called when it is the player's turn to act.
- `public void` **`gameStartEvent`**`(GameInfo gInfo) {`
  A new game has been started.
- `public void` **`stageEvent`**`(int stage) {`
  A new betting round has started.
- `public void` **`showdownEvent`**`(int seat, Card c1, Card c2) {`
  A showdown has occurred.
- `public void` **`actionEvent`**`(int pos, Action act) {`
  An action has been observed.
- `public void` **`gameOverEvent`**`() {`
  The hand is now over.

4.3 Pre-Flop Betting Strategy

The betting strategy used in pre-flop is relatively simple, and is based in two parts: the assessment of the initial hand of the agent, and an expert system that selects a strategy to use during pre-flop.

4.3.1 Initial Assessment

The assessment of the *hole cards* of HuBot is done by using Income Rate tables. These tables define a value for each distinct hand before the flop. While these values do not necessarily reflect an accurate estimate for the expect value of the hand, they do provide a first-order approximation. This topic has been described in Section 3.2.1.1.

There are three Income Rates tables loaded into memory in the beginning of a match. These tables were created taking into account the number of opponents, and so, there is one table for *heads-up*, one table for a group of three or four players, and one table for five or more players.

	2	3	4	5	6	7	8	9	T	J	Q	K	A
2	-6	-462	-422	-397	-459	-495	-469	-433	-383	-336	-274	-188	-39
3	-180	21	-347	-304	-365	-418	-447	-414	-356	-308	-248	-163	-1
4	-148	-69	67	-227	-273	-323	-362	-391	-334	-287	-223	-133	32
5	-121	-38	31	122	-198	-230	-270	-303	-309	-259	-200	-103	64
6	-174	-95	-10	64	206	-151	-175	-204	-217	-235	-164	-72	23
7	-208	-135	-47	35	108	298	-87	-106	-112	-128	-124	-26	72
8	-184	-164	-83	2	93	168	420	-5	6	-10	-10	22	126
9	-146	-128	-111	-26	64	153	245	565	134	118	118	151	189
T	-88	-68	-46	-29	59	155	268	383	765	299	305	336	373
J	-38	-15	1	30	51	147	256	377	536	996	380	420	462
Q	35	49	72	99	127	162	268	384	553	628	1279	529	574
K	117	141	167	190	223	261	304	423	591	669	764	1621	712
A	269	304	333	363	313	365	416	475	644	720	815	934	2043

Table 3: Income Rate table for 7 players.

In the table above, each *cell* defines a distinct hand. Suited hands are shown in the bottom left part of the table, whereas unsuited hands are shown in the top right triangle of the table.

HuBot looks up in one of these tables the value of its initial hand, selecting the table based on the number of expected players that will see the flop. The number of expected players is calculated this way:

$$\begin{aligned}&\boldsymbol{expected\_num\_players} = num\_guaranteed \\ &+ probability\_play \times (active\_players - num\_guaranteed) \qquad (4)\end{aligned}$$

- ❖ ***num_guaranteed*** is the number of players that have already put money into the pot, and therefore committed themselves to play the hand. This includes the blinds and HuBot (the agent considers that he will play the hand).
- ❖ ***active_players*** reflects the number of players still active in this hand, including the players already committed.
- ❖ ***probability_play*** is a fixed value, selected *ad hoc*. It defines the probability of a player to play his hand in the preflop. This value is currently set to 0.4, after empirical testing.

Succinctly, this formula can be read as: the expected number of players is the number of players already committed (and blinds) plus each remaining player that hasn't acted yet times 0.4.

The *expected_num_players* variable is then rounded to nearest integer and falls into one of the groups of players: *group.**TWO***, *group.**THREEORFOUR***, *group.**FIVEPLUS***.

Afterwards, the Income Rate table used is selected, based on the group decided, and a value is retrieved representing the expected value of the agent's hand.

4.3.2 Choosing a strategy

There are six possible strategies to choose from in the preflop: **Make0**, **Call1**, **Make1**, **Call2**, **Make2**, and **Make4**. These strategies are sequential rules defined by a poker expert, and are described in the next Section in more detail.

One strategy is selected when it is HuBot's first time to act in the pre-flop. After a strategy has been selected, it is used for all further decisions in the pre-flop.

In order for HuBot to choose a strategy, threshold values are assigned to the six strategies, giving lower values to the passive strategies and higher values to the aggressive strategies. These threshold values define the minimum value that a hand must have in order to play the hand with this strategy. For example, **Make4** strategy, which tries to raise the pot every time, might have a threshold value of 900, forcing HuBot to only play this strategy when it has a very high pocket pair, or a hand such as Ace-King suited.

Thresholds have a base value, but also vary according to the expected number of players and the position of the agent in the table. The position is defined by the proximity to the dealer in this hand (i.e. the dealer is in ***position*** zero, the player on the right of the dealer is in ***position*** one, and so on...).

The linear formulas used to compute the thresholds have been described in detail in [7] and the threshold values have been proposed by the professional poker player Darse Billings.

4.3.3 Rule-based strategies

These strategies share the name of Loki-1 strategies, but have little in common, as they implement a more sophisticated system. A small overview of the six possible strategies is given below:

- **Make0**: Usually fold, except when the agent is in the blinds.
- **Call1**: Calls one bet.
- **Make1**: Calls one bet in late position. On rare occasions raises.
- **Call2**: Can call as much as two bets.
- **Make2**: Raises up to two bets. On rare occasions folds.
- **Make4**: Tries to cap the pot.

Each strategy contains expert knowledge, and handles different case scenarios. These scenarios are distinguished based on several context variables, such as:

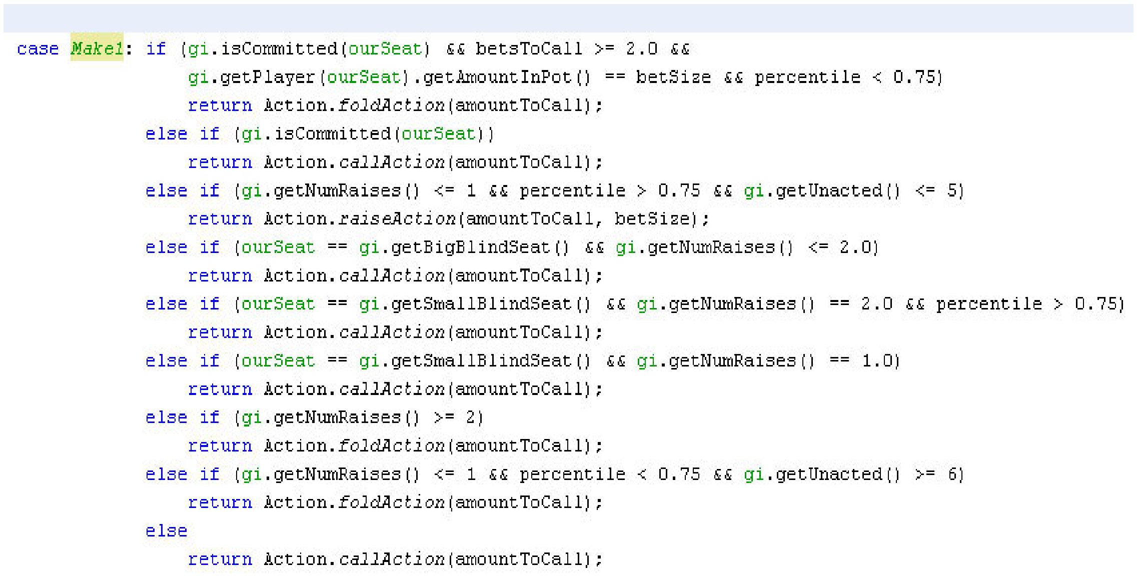
- how much money has the agent already committed into the pot;
- how many bets is the agent facing, in order to call;
- number of players yet to act in this round;
- the ***'percentile'*** variable.

The ***percentile*** represents how close the value of the agent's hand is to the next strategy's threshold value. It is used by some strategies to smooth the thresholds, deciding upon different actions based on the value of the hand. For example, let's consider that HuBot is playing the hand A♠J♠, which has an expected value of 720. The thresholds for the strategies **Make2** and **Make4** were respectively 480 and 900, thus **Make2** strategy was selected. ***percentile*** is calculated this way:

$$\boldsymbol{percentile} = \frac{720-480}{900-480} = \mathbf{0.571} \qquad (5)$$

In this example, since the ***percentile*** is above 0.49, if HuBot is faced with a scenario when no player has raised yet, it will raise. However, in the same scenario, with a lower percentile, HuBot might simply check.

Furthermore, in these strategies' rules, special treatment is given if HuBot is playing the small blind or big blind.



```
case Make1: if (gi.isCommitted(ourSeat) && betsToCall >= 2.0 &&
                gi.getPlayer(ourSeat).getAmountInPot() == betSize && percentile < 0.75)
                return Action.foldAction(amountToCall);
            else if (gi.isCommitted(ourSeat))
                return Action.callAction(amountToCall);
            else if (gi.getNumRaises() <= 1 && percentile > 0.75 && gi.getUnacted() <= 5)
                return Action.raiseAction(amountToCall, betSize);
            else if (ourSeat == gi.getBigBlindSeat() && gi.getNumRaises() <= 2.0)
                return Action.callAction(amountToCall);
            else if (ourSeat == gi.getSmallBlindSeat() && gi.getNumRaises() == 2.0 && percentile > 0.75)
                return Action.callAction(amountToCall);
            else if (ourSeat == gi.getSmallBlindSeat() && gi.getNumRaises() == 1.0)
                return Action.callAction(amountToCall);
            else if (gi.getNumRaises() >= 2)
                return Action.foldAction(amountToCall);
            else if (gi.getNumRaises() <= 1 && percentile < 0.75 && gi.getUnacted() >= 6)
                return Action.foldAction(amountToCall);
            else
                return Action.callAction(amountToCall);
```

Fig 8: Rules of strategy Make1.

Figure 8 shows the rules of one of the strategies. The algorithm starts by evaluating rules that consider if the agent is already committed. If not, then other rules are evaluated, from the most aggressive to the most passive. In the end, if no rule was selected, then a default action is taken. In the case of Make1, the default action is to Call.

All the other strategies follow this same structure.

These rule-based strategies have been adjusted throughout HuBot's version. The most extensive improvement took place in version 1.08. Since then, results are very satisfactory regarding the frequency of hands played in the pre-flop. The percentage of flops seen for HuBot is around 22%, which is the percentage suggested by Poker Academy Pro, as shown in Figure 17, in Chapter 5.

Before the flop, *weight tables* aren't taken into account when HuBot chooses an action to take, but are, nonetheless, updated whenever an action from a player is observed. This re-weighting system is explained in detail throughout section 4.5.1.

4.4 Post-Flop Betting Strategy

After the flop, HuBot chooses an action by evaluating its hand in relation to the board, and by using a set of betting rules and formulas to translate this value into an action. HuBot's beliefs about the hands its opponents might hold are accounted for in this phase of the game, and greatly influence the agent's decision.

4.4.1 Hand Evaluation

For evaluating a hand in post-flop, HuBot uses the Hand Strength (HS) and Hand Potential algorithms, described in Section 3.2.1.

However, against multiple opponents, the Hand Strength algorithm results have a degree of error [7]. This error comes from the fact that the algorithm doesn't handle each opponent independently, and so ignores the interdependencies that arise from the fact that two players cannot hold the same card. For example, situations where one player is holding A♡Q♡ and another is holding Q♡J♠ should not be considered in the calculation, since they are naturally impossible to occur.

In order to treat each opponent independently, the algorithm would need an extra iteration layer for each opponent. Each additional iteration layer increases the computational complexity by a factor of about 1000, making it unfeasible to compute with just three players.

The method of extrapolating Hand Strength against multiple opponents, described before (Eq.3), can be replaced by a better approach, in order to minimize the average error of this estimation.

In [9], one abstraction was proposed, called Expected Hand Strength Squared – $E[HS]^2$. This method is based in the Sum of Squares approach and measures the expected value of the square of the hand strength. In practical terms, the goal of $E[HS]^2$ is make the better hands carry more weight in the calculation than the weaker hands.

This idea has been extended to introduce a new approach – $\mathbf{EHS_n}$. Instead of squaring the winning percentage of every hand (HS), we exponentiate this value to the power of the number of opponents. The final result is calculated by adding the exponentiated values for every possible board cards in the *River*.

Figure 9 shows an un-weighted EHS_n calculation, done in the flop with a two-card look-ahead.

```
HandStrength(ourcards,boardcards)
{
   totalBoards = 0
   /* All possible board cards to come. */
   for each case(turn)
   {
       for each case(river)
       { /* Final 5-card board */
           board = [boardcards,turn,river]
           ourrank = Rank(ourcards,board)
           winds = ties = showdowns = 0
           totalBoards++

             /* Consider all two-card combinations of the remaining
             cards for opponent. Excludes the board cards.*/
             for each case(oppcards)
             {
                 opprank = Rank(oppcards,board)
                 if(ourrank>opprank)           wins += 1
                 else if(ourrank==opprank)     ties += 1
                 else                     showdowns += 1
             }
             hs = (wins + (ties/2)) / showdowns
             EHSn += hs ^ nr_opponents
       }
   }
   EHSn = EHSn / totalBoards
   return EHSn
}
```

Fig 9: Raw EHS_n calculation in the flop.

In practice, HuBot considers the *weight tables* of each opponent when calculating the EHS_n, thus calling it ***wEHS**$_n$*. This calculation is done simply by (a) adding an iteration layer per opponent, of the nested *for* cycle that enumerates all opponent hands; and (b) adding weights to the possible opponent's hands, instead of incrementing it.

With this new method, the average error is much lower than with the previous approaches (see Figure 10).

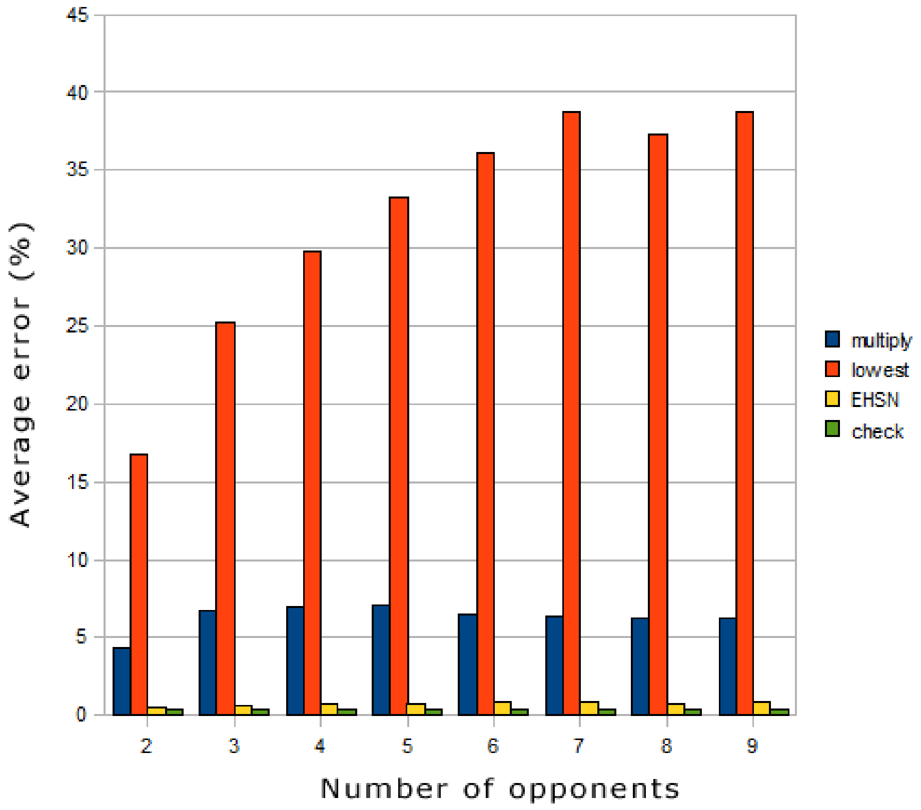


Fig 10: Comparison of multiple opponent Hand Strength extrapolation methods.

In Figure 10 the average error (in percentage) of three different Hand Strength Extrapolation methods is represented. The blue bars represent the method of extrapolating HS by multiplying the HS of each opponent together, suggested in earlier papers of the CPRG, which formula is explicit in Eq.3. The red bars regard a very naive method that consists in simply using the HS from the hardest opponent (the one with the best winning percentage against our hand). The yellow bars are the average error of the EHS_n method.

These results come from testing these methods against a Monte Carlo simulation, based on several thousand hands and flop cards chosen randomly. Thus, it also holds a degree of error by itself, represented in the green bars. The average error of the Monte Carlo simulation was of 0.4%, and it also reflects the accuracy of this test.

One important observation is that the yellow bar never exceeded the 1% average error.

Another variable HuBot estimates is the Positive Potential (**PPot**), calculated using the Hand Potential algorithm (see Section 3.2.1).

The algorithm described was simply adjusted to account for *weight tables*. Regarding the considerations of multiple opponents in the Hand Potential calculation, it is believed that the value of Hand Potential against one player is a simple but reasonable estimate [7].

Therefore, before calling the Hand Potential function, the agent combines the *weight tables* of all the active opponents into one unique *table*, or in other words, averages the probability distribution over the possible hands for his opponents. This new table is called a *field array*, and is then used to calculate the Positive Potential for HuBot's cards, storing the result in a variable called ***faPPOT***.

In the Hand Strength and Hand Potential algorithms, there is a function called within to rank the hands. The goal of this function is to assign to all possible hands different values, such that better hands have higher value, and worse hands have lower value. This value is then used to compare hands, and easily determine which one is the best.

While ranking functions can be used to rank 5-card hands and 6-card hands, it's more common in Texas Hold'em to use them for ranking 7-card hands, which consist of the 2 cards of the player and the final 5 cards at the River.

In the algorithm of EHS_n (Figure 9), we can see that the ranking function is called exactly 1,071,271 times:

$$\binom{47}{2} + \binom{47}{2} \times \binom{45}{2} = 1071271 \qquad (6)$$

This includes ranking 1,070,190 opponent's hands, within the inner loop of the algorithm.

$$\binom{47}{2} \times \binom{45}{2} = 1070190 \qquad (7)$$

Thus, the speed of this function is of great importance. Throughout the years, many developers came up with new hand ranking algorithms, consistently trying to improve the efficiency. These algorithms have evolved a lot since CPRG ranking function, which had an average ranking speed of 1,165,716 hands per second.

Currently, the state of the art algorithm, which implements the fastest function, is the *RayW Look-Up Tables Hand Evaluator* [17], which is able to evaluate an average of 142,779,680 hands per second. However, the current implementation of this algorithm suffers some limitations, as it uses about 128Mb of memory, and is only available for ranking 7-cards hands.

Since HuBot requires a java algorithm, able to rank also 5-cards hands, it is currently using *Steve Brecher's Hand Evaluator* [36], which can rank about 34,659,212 hands per second.

These speed results have been calculated experimentally, by enumerating all possible 7-cards hands, and ranking each and every one of them.

However, there is some redundancy in ranking 1,071,271 hands (Eq.6). In fact, in the Flop stage, there are only four unknown cards to account for: two board cards yet to come, and the two cards the opponent holds. So, there are only $\binom{47}{4} = 178365$ unique rankings for a specific flop board and the agent's hand. This means that the agent should rank exactly this number of hands, and not more.

To account for this, HuBot implements a *pre-calculation table*, where the ranks of the 178,365 hands are calculated in the beginning of the flop, and stored in this table for the rest of that game. Therefore, this table substitutes the ranking function, as the values can be obtained by simply looking up in the table.

The result is a considerable improvement in speed, in the calculations of Hand Strength and Hand Potential throughout the program.

However, the inner loop is still executed 1,070,190 times (Eq.7).

4.4.2 Formula-based system

After evaluating his hand, HuBot decides upon its action based on a set of formulas and rules, by using the calculated variables $\boldsymbol{wEHS_n}$ and ***faPPOT***. In fact, the agent bases his decision on one of six possible strategies (e.g. betting based on the strength of his hand).

Each strategy has a priority, determining the order in which the strategies are considered. Higher priority is given to strategies such as check-raising and semi-bluffing, whereas passive strategies such as calling based on showdown odds has the lowest priority. Whenever a strategy isn't chosen, the strategy with the next highest priority is considered.

Before its first action in the flop, HuBot calculates a variable called *aggressiveness factor* that will influence the way he plays his hand until the rest of the game. This variable is a number chosen randomly from 0 to 99.

The six possible strategies in the post-flop are presented below, from the one with the highest priority to the one with the lowest:

1. Check-raise

A *check-raise* strategy consists in checking as the first action, with the intention of raising in the same betting round after an opponent bets. Therefore, this strategy should be used when we hold a very good hand.

HuBot only uses this strategy maximum one time per game, and only if the *aggressiveness factor* is more than 0.74. So, if HuBot hasn't employed this strategy before, he will use it in 25% of the following situations:

- When his Hand Strength (***wEHSn***) is superior to 0.85, he's sitting in an earlier position of the table, and there are at least two more players to act after HuBot.

The two last considerations are done to ensure that there is a good probability that someone might bet after the agent checks.

By only using it 25% of the times, it adds unpredictability to the agent's playing style, causing his actions to have a dual interpretation, and a harder task for his opponents to infer a conclusion with certainty.

2. Based on strength

If HuBot can't check-raise, then the next strategy in line to be considered is to bet/raise or call based on his Hand Strength.

The first step of this strategy is to check if the $\boldsymbol{wEHS_n}$ of the agent's hand is higher than 0.70. If so, HuBot bets or raises. However, if his Hand Strength is between 0.50 and 0.70, the agent acts with more cautious. In this scenario, the agent's goal is to make one bet, and so, if someone else already betted before him, he will simply call the bet.

In the cases that he is facing zero bets to call, HuBot will bet himself. However, with an exception: 50% of the times that he is in an earlier position with two or more opponents yet to act, he will simply call. This is done because HuBot predicts with a high probability that someone else will already bet.

3. Semi-bluff

Semi-bluffing is a strategy that consists in betting, in the Flop or Turn, with a hand that has a good chance of winning by the showdown. This strategy has been successful in earlier versions of Poki.

It is based on the Hand Potential estimation. If HuBot's hand has enough chance to improve enough to call for a bet and a raise, then he will open the betting himself.

Semi-bluffing is done when ***faPPOT* > *pot_odds2***, where:

$$potodds2 = \frac{2 \times bet\_size}{pot\_size + 6 \times bet\_size} \quad (8)$$

This equation represents the money that we have to put on the pot ('$2 \times bet\_size$' is our bet, and possibly calling a raise), in proportion to the money that we can win (the current money in the pot plus an estimation of the bets in this round).

The goal of this strategy is to bluff other players to fold after our bet. However, if that doesn't work, our hand still has a good chance of winning.

One restriction that HuBot implements for employing this strategy is that he must be in a late position, and only playing against one or two opponents. This has been established, after observing that against many opponents, this strategy doesn't perform so well.

In case no one raises his bet, HuBot will continue to bet (only to open the betting) in the subsequent rounds even without enough Hand Potential, as there is a reasonable chance of winning the pot immediately. This is known as a **continuation bet**.

4. Bluff

Bluff is a complex strategy that HuBot considers when it is up against only one opponent.

This strategy takes into account several parameters, and for a positive scenario, it is only used 50% of the times. The scenario required for employing *bluffing* is very specific, and can be divided in two rules:

- The agent is up against only one opponent, hasn't used this strategy before in this game, and *pot odds* for the opponent must be worse than 7 to 1.
- The predicted probability that the opponent will fold when faced against HuBot's bet must be higher than HuBot's *pot odds* and not inferior to 0.20.

In poker, *pot odds* are the ratio of the current size of the pot to the cost of a contemplated call, and its calculation will be explained during the description of the next strategy.

The first rule establishes a maximum of the pot odds for the opponent. This is done, because if the pot is big enough, an opponent might call us with virtually any hand, solely based on his pot odds.

The second rule determines the probability of the opponent folding immediately after the agent's bet, by using the Opponent Modeling, and getting this value from an

Action Frequencies table (described in Section 4.5.2). It also ensures that the agent's *pot odds* are good enough to cover for the opponent's probability of folding. The reason behind this is that, if the opponent only folds very rarely in a specific situation, then HuBot should only *bluff* if he expects to have a good return of investment (good *pot odds*), knowing that usually he'll get his *bluff* called.

Due to the difficulty of testing the expected value of this strategy experimentally, it has only been tested empirically so far. The results have been very satisfactory, making *bluffing* a very profitable strategy, and also a good way to add unpredictability to the agent's play. In fact, since HuBot is now able to *bet* with any possible hand, it makes it harder for the opponents to draw accurate conclusions from his actions.

5. Pot odds

If none of the above strategies were selected, it means that HuBot's Hand Strength is less than 0.50.

In case no one else before HuBot betted (zero bets to call), then the agent will simply *check*. However, when faced with bets to call, he will try to call based on *pot odds* and if it fails, based on *showdown odds*.

'Pot odds' is the proportion between our winning chances to the expected return from the pot.

Let's consider an example: HuBot is in the River with a 20% chance of winning. His only opponent bets \$4, making the pot contain \$20. If the agent calls in this situation, he is expected to lose 4 times out of 5, losing \$4 each time. However, 1 time out of 5, he's expected to win \$20. Thus, in the long run, based on his estimation of the percentage of winning, this decision will be profitable.

The formula for calculating *pot odds* is defined below:

$$potodds = \frac{amountToCall}{pot\_size + amountToCall} \tag{9}$$

The concept for calling based on *pot odds* is slightly different. On the *River*, HuBot simply verifies if his Hand Strength ($\boldsymbol{wEHS_n}$) is higher than the *pot odds*. However, in the Flop and Turn, there are still more cards to come, and so, the decision is based on the Hand Potential. If the condition '$faPPOT > potodds$' is accepted, then the agent will call. This considers that if the probability of getting a strong hand with the next card is higher than the current *pot odds*, then the profitable decision is to call.

This strategy is very useful when we're holding a *drawing hand* (see appendix A).

6. Showdown odds

'Showdown odds' is the last strategy considered to call for bets.

It is a strategy considered only in the Flop and Turn stages, and addresses situations when the agent's hand is strong enough to show profit by the showdown, therefore discouraging frequent *bluffing* by the opponent. This defensive strategy has been designed because calling based on *pot odds* only considers the immediate potential to improve.

Showdown odds are calculated in separate ways for the Flop and for the Turn. On the Turn:

$$showdownOdds = \frac{amountToCall + bet\_size}{pot\_size + amountToCall + 2 \times bet\_size} \qquad (10)$$

This formula is similar to the formula for the *pot odds*, except that it considers one more bet that the agent will call in the next stage, the River.

The formula for the Flop is the following:

$$showdownOdds = \frac{amountToCall + 2 \times 2 \times bet\_size}{pot\_size + amountToCall + 2 \times 4 \times bet\_size} \qquad (11)$$

Simply, it expects that the agent will call one more bet in the Turn and one more bet in the River stages. In Limit Hold'em the bet size doubles when going to the Turn, and this is way the '*bet_size*' is also doubled in the formula.

Afterwards, the agent compares his Hand Strength at showdown ($\boldsymbol{wEHS_n}$) with the *showdown odds* in the form '$wEHS_n > showdown\_odds$', and if it's higher then he will call.

Finally, if this strategy isn't also accepted for the current game situation, HuBot will simply fold his hand.

Conclusion

The limits used in the strategies above have been established in an *ad hoc* manner, by a poker expert, and tested empirically.

Furthermore, these strategies are used, in the exact described form, whenever it is HuBot's first action in a betting round. If it is subsequent action, slight changes are done to these strategies, depending on whether the first action was a check or a bet. These changes are not described in this document, as they are not significant.

With these six strategies, HuBot attempts to address important gameplay concepts in poker, such as **deceitfulness** and **unpredictability**. By doing so, it makes it much more difficult for an opponent to accurately model his playing style, and subsequently exploit it.

4.5 Opponent Modeling

The algorithms described for Hand Strength and Hand Potential are unbiased, by assuming that the opponent is equally likely to hold any possible two-card combination.

As suggested before, small changes can be done to these algorithms in order to include a probability distribution over the possible hands. This is done in HuBot by using *weight tables*. This topic has been introduced in the end of Section 3.2.1.

HuBot maintains a *weight table* for every player, including himself, and updates it after a player's action. This is process is called **re-weighting**.

A *weight table* consists of assigning values to each possible hand the opponent can hold. These values represent the conditional probability of the opponent having played that hand to the current point of the game. This can also be understood as the agent's belief that the opponent is likely to hold such cards, considering his past actions in the game.

The entries in the *weight table* vary from zero to one, rather than absolute probabilities, since the calculations for Hand Strength and Hand Potential require relative probabilities. In the beginning of each new game, all the values in the *weight tables* are set to one.

4.5.1 Re-Weighting

Weights are updated both after cards are dealt and after a player's action.

After HuBot receives its *hole cards*, many hands become impossible for the opponent's to hold, and so, the weights for these hands are set to zero. The same happens whenever public board cards become known.

Updating the weights after a player acts is more complex task, and is related with the "***`actionEvent(int pos, Action act)`***" event of the Meerkat API.

In order to update the weights for a particular action, the agent needs to predict the probability that the opponent would make that action in the current situation. Predicting an opponent's action is described in the next Section: **Action Frequencies Tables**.

This is because re-weighting is based on the threshold hand value needed for the possible actions, and these thresholds come from the probability that the opponent folds, check/calls, and bet/raises in the exact same situation.

The re-weighting function used is different, depending if the player check/called, or bet/raised. No re-weighting is done in case a player folds (since his *weight table* won't be used more during the current game).

First, we'll start by explaining the re-weighting function for the action bet/raise.

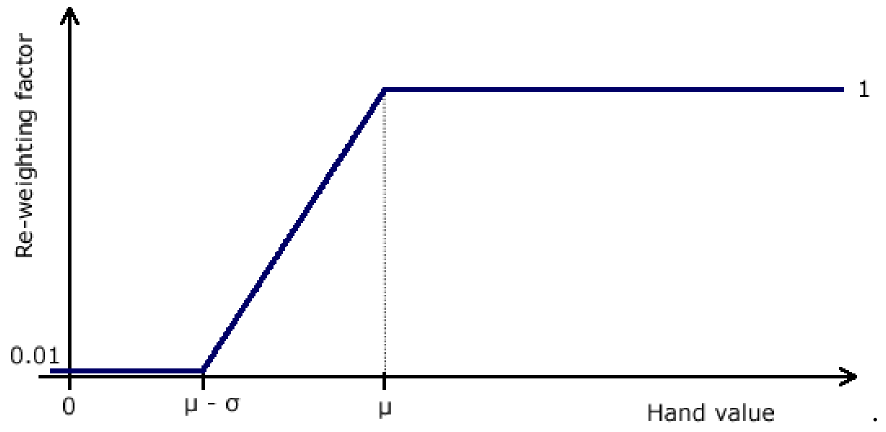


Fig 11: Re-weighting function for a bet/raise.

This re-weighting function is based on the premise: considering that the player raised, and in this situation he usually only raises 20% of the times, then he must have one of top 20% hands. Therefore, the top 20% hands should remain with their weight in the *weight table*, and the other hands should have their weight set to zero.

The function on Figure 11 assigns a re-weighting factor to each hand, depending on its hand value. Afterwards, the re-weighting factor is used to update a weight in the *weight table* by multiplying it against the current weight. That means that a re-weighting factor of one keeps the weight for that hand the same.

Furthermore, the weight of one hand is not allowed to go under 0.01, since we don't want to exclude any possible case.

Some considerations about the function in Figure 11:

μ – Denotes the threshold needed for the action 'raise'. This is based on the estimated probability of raise by the player in this situation. If, for example, we estimate a probability of raising of 20%, then **μ** should be the hand value of the last hand in the top 20% hands.

So, in order to get **μ**, a sorted list of the hand values is created for all possible hands the player might hold. In the pre-flop stage, these hand values correspond to the Income Rates of the hands, and are retrieved from the table shown in Table 3. In the post-flop, a simple way to get the threshold would be to simply subtract the raise probability from one, and use the resulting value directly. However, this method has been proved to be error-prone in [1].

Therefore, for the post-flop, HuBot also creates a sorted list of hand values. These values are obtained by calculating the (weighted) Hand Strength and Hand Potential for each hand. Since this means an expensive computation, these algorithms only perform a 1-card look-ahead. The hand value assigned to each hand is then a combination of the Hand Strength and Positive Potential values, as in Eq.3.

In HuBot's implementation, these sorted lists of hand values are stored in objects '***ValueSorting***'.

σ – Represents the uncertainty we have about our estimation of the raising probability of the player for this situation. This extends the premise above, by adding an interval to account for this uncertainty. In the interval $[\boldsymbol{\mu} - \boldsymbol{\sigma}, \boldsymbol{\mu}]$ the function interpolates linearly between **0.01** and **1**, according to the formula:

$$reweight\_factor = \frac{hand\_value - \mu + \sigma}{\sigma} \tag{12}$$

In the pre-flop, the value of **σ** has been selected in an *ad-hoc* manner. It is established at **330**, since 68.26% of the hands lie in the Income Rate range -323 to +336.

For the post-flop, **σ** is calculated by the formula:

$$\sigma = 0.4 \times (1 - \mu) \tag{13}$$

By making **σ** vary with **μ**, we manage to address the tendency that *loose* players (players with a low **μ**) exhibit more uncertainty, whereas *tight* players (high **μ**) tend to be more consistent with the threshold.

Another way to account for uncertainty is to establish a maximum limit to **μ**. This is done to prevent cases when **μ** is getting too close to the maximum hand value in the sorted list. HuBot defines this maximum value as the hand value of the last hand in the top 10% of the sorted list.

The re-weighting function for the action check/call is slightly more complex:

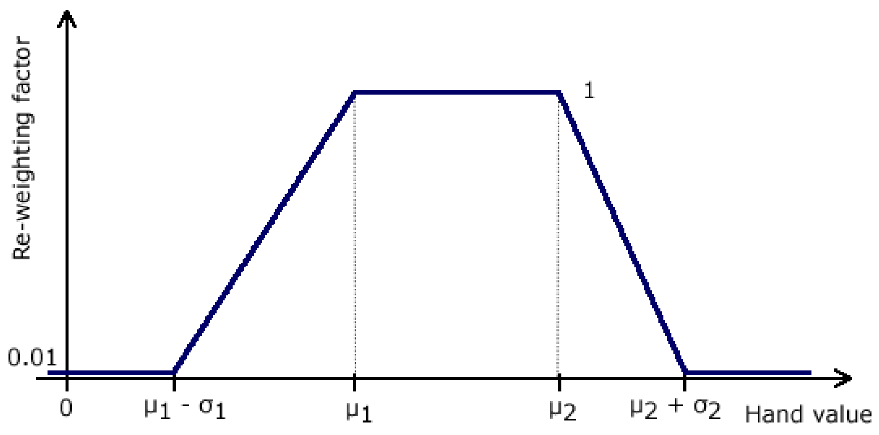


Fig 12: Re-weighting function for a check/call.

When a player calls, we get the threshold for the action 'call' and the threshold for the action 'raise'. These are defined in Figure 12 as μ_1 and μ_2 respectively. For example, if a player was observed in this situation to fold with a probability of 30%, then μ_1 would be the hand value for the last hand in the top 70% of the sorted list.

The thresholds are obtained from a sorted list of hand values, exactly like explained before, and the calculation for σ_1 and σ_2 is also similar.

As we can see from the Figure 12, this graph consists of three main branches. The first one interpolates linearly in the interval . The hands with hand value between μ_1 and μ_2 are in the second branch of the graph, and are not re-weighted. The third branch has the domain and interpolates linearly between **1** and **0.01**. All the other hands will be re-weighted by a factor of **0.01**.

To account for some uncertainty in the estimations, μ_1 is limited to a maximum of top 10% hand values and μ_2 is limited to a minimum of top 90% hand values.

One important consideration is that when re-weighting a check/call decision, the player might have a *drawing hand* and base his decision solely on *pot odds*. In these cases, the re-weighting function is unable to re-weight the hand accurately, since it is considering also the Hand Strength of the hand.

In order to address these hands with a low Hand Strength and a very high Hand Potential, an adjustment was introduced: when the Positive Potential of hand is higher than the *pot odds* (Eq.9) then the weight for that hand will not be reduced.

This way, the hands with a good *draw potential* are not longer underestimated.

As a final note, the *weight tables* are only updated once per player per round. A copy of the *weight table* is saved in the beginning of the round, and each time a player takes an action that has a higher threshold, the re-weighting is performed in the saved *weight table*. For example, if a player checks, and later in that round is seen to raise, the threshold value for re-weighting will be higher, and so requires a new re-weighting.

In this way, it is possible to adjust the weights correctly when a player uses a *check-raise* strategy.

However, if a player, intending to employ this strategy, checks in his first action and all the other players check as well, he will not be given the chance to raise in the same betting round. As a result of this, the re-weighting function will decrease the weights of very good hands, when in fact he might be holding one of these hands.

To account for this, a minimum re-weighting factor is established for the hands with a hand value better than $\boldsymbol{\mu_2}$. This value is exactly equal to our prediction probability that the player will raise, if given the opportunity, in the current round.

In this way, even if a player fails to complete his *check-raise* strategy, the reduction of weights for the best hands will not be so drastic.

4.5.2 Action Frequencies Tables

HuBot implements a statistical-based Opponent Modeling, by keeping *action frequencies tables* for his opponents. This type of Opponent Modeling has been introduced in Section 3.2.2.

The tables store the number of times a player folds, calls or raises, in a given situation. These frequencies are used in HuBot for computing the thresholds in the re-weighting system and for predicting an opponent's action in some parts of the program. An example of this is when HuBot considers *bluffing* in the post-flop by predicting that the opponent will fold if he bets.

Each entry of the *action frequencies table* defines a possible context (or situation).

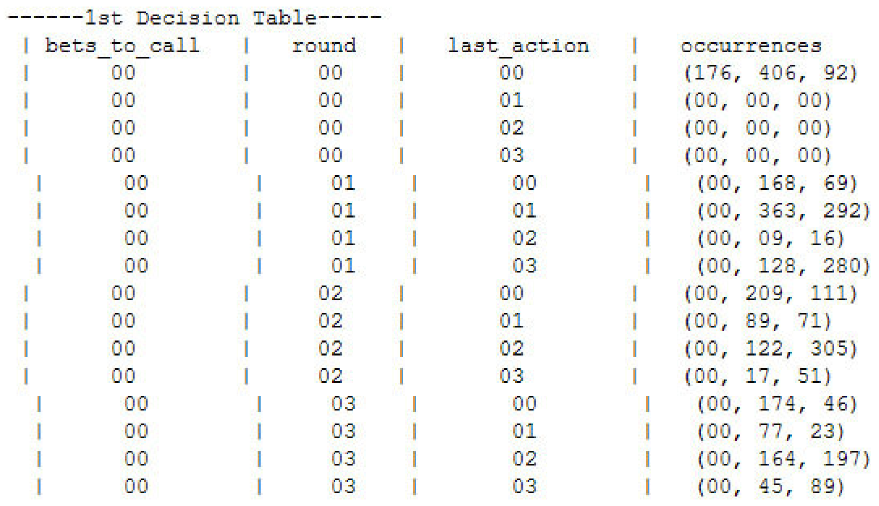
HuBot maintains two such tables per player.

The first table concerns only the first action of a player in a betting round. There are 39 distinct contexts in this table, categorized by the three following properties: **bets to call**, **round**, and **last action**.

Bets to call represents the number of bets necessary to call, and has three possible different values: **{0, 1, 2+}**. *Round* separates between the different betting rounds, and thus can assume four different values representing **{pre-flop, flop, turn, river}**. *Last action* represents the action taken by the player in the last betting round. It considers four different actions: **{check, call, bet, raise}**.

With these three properties, the contexts are broad enough to capture the essence of the opponent's play. The number of entries in the table are: $3 * 4 * 4 = 48$. However, some of these entries do not define a valid context: in the pre-flop, the variable *last action* is not used, since pre-flop is the first betting round. Therefore, these entries of the table are kept with the value zero.

The choice to use these context variables was based on the results obtained in [6] and [13], after using an ANN for Opponent Modeling.



```
------1st Decision Table-----
| bets_to_call   |    round    |    last_action    |    occurrences
|      00        |     00      |        00         |    (176, 406, 92)
|      00        |     00      |        01         |    (00, 00, 00)
|      00        |     00      |        02         |    (00, 00, 00)
|      00        |     00      |        03         |    (00, 00, 00)
|      00        |     01      |        00         |    (00, 168, 69)
|      00        |     01      |        01         |    (00, 363, 292)
|      00        |     01      |        02         |    (00, 09, 16)
|      00        |     01      |        03         |    (00, 128, 280)
|      00        |     02      |        00         |    (00, 209, 111)
|      00        |     02      |        01         |    (00, 89, 71)
|      00        |     02      |        02         |    (00, 122, 305)
|      00        |     02      |        03         |    (00, 17, 51)
|      00        |     03      |        00         |    (00, 174, 46)
|      00        |     03      |        01         |    (00, 77, 23)
|      00        |     03      |        02         |    (00, 164, 197)
|      00        |     03      |        03         |    (00, 45, 89)
```

Fig 13: A first Action Frequencies table, represented partially.

Figure 13 shows the state of an Action Frequencies table for the first action of a player, after thousands of games have been played against this player. In the figure, there are only shown the contexts with zero bets to call, therefore, only the first 16 entries of the table.

In order to be able to adapt to the opponent's changing strategies, if there are more than 30 observations for a particular context, then HuBot gives more weight to the last 30 observations, when calculating the probability of an action taken place. This permits HuBot to track better moving targets – players that change their playing style over time.

When playing against a new opponent, HuBot doesn't have the convenience of a table filled with many observations. So, whenever there are less than 30 observations for given context, the agent combines the table entries in order to generate predictions. This technique works by merging similar contexts.

Considering this, the table seen in Figure 13 is organized from the most significant and general variables to the most specific. This means that, firstly contexts are grouped by the same *round*, and then, if still there aren't 30 observations, they are grouped by the same *bets to call*.

In this way, it is possible to spot general trends of an opponent's play with much fewer games played.

In the case there aren't 30 observations in the same *bets to call* group, then some default values are used, based on the *bets to call* variable:

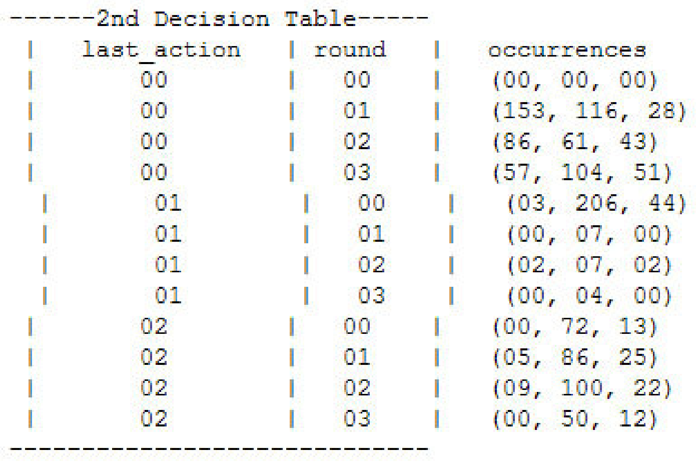
bets to call	Fold	Check/Call	Bet/Raise
0	0	0.7	0.3
1	0.5	0.3	0.2
2+	0.7	0.2	0.1

Table 4: Default frequency values for the first Action Frequencies table.

These values are a crude estimation of the action frequencies for the average player, and were calculated by observing Action Frequencies tables of many players.

Subsequent actions of a player in the round are addressed by a second *action frequencies table*. This is because a second action in the round is better predicted by taking into account another context variable: the action taken in the first round of the round, defined here also as ***last action***. This variable has the domain {**check**, **call**, **bet/raise**}.

This way it is possible to observe the number of times an opponent successfully performs a check-raise strategy. Modeling these types of complex strategies is important to make profitable decisions throughout the game.



```
------2nd Decision Table-----
 |   last_action    | round   |   occurrences
 |       00         |   00    |   (00, 00, 00)
 |       00         |   01    |   (153, 116, 28)
 |       00         |   02    |   (86, 61, 43)
 |       00         |   03    |   (57, 104, 51)
  |       01         |   00    |   (03, 206, 44)
  |       01         |   01    |   (00, 07, 00)
  |       01         |   02    |   (02, 07, 02)
  |       01         |   03    |   (00, 04, 00)
 |       02         |   00    |   (00, 72, 13)
 |       02         |   01    |   (05, 86, 25)
 |       02         |   02    |   (09, 100, 22)
 |       02         |   03    |   (00, 50, 12)
-----------------------------
```

Fig 14: A second Action Frequencies table.

Figure 14 shows an Action Frequency table that keeps track of the second decision of a player in the betting rounds.

This table has also been constructed from the most significant context variable to the most specific. In this case, contexts with the same *last action* can be merged to obtain a prediction.

The default frequencies of this table are shown in Table 5.

last action	Fold	Check/Call	Bet/Raise
Check	0.4	0.4	0.2
Call	0.1	0.8	0.1
Bet/Raise	0.1	0.6	0.3

Table 5: Default frequency values for the second Action Frequencies table.

Every time a player acts, these Action Frequencies tables are updated, considering whether it is the first action of that player in the round or the second. In the case it's a subsequent action, these tables are not updated.

However, the thresholds for subsequent actions used in the re-weighting functions are still derived from the second Action Frequencies table. This consideration is due to the fact that the playing style of a player after his second decision is usually coherent with his second decision. Not many strategies arise from the third action of a player in the same betting round.

As a final note, whenever HuBot faces an opponent that he played against before, he loads into memory the previously saved Action Frequency tables for that player. To do so, HuBot always stores in files these tables for his opponents, after finishing a game.

The results of HuBot's statistical-based **Opponent Modeling** are very satisfactory, and can be seen under the next chapter.

Chapter 5

5. Assessment of Results

The assessment of the performance in poker is a difficult task, since the element of luck dominates the outcome of any game.

Nonetheless, it is possible to measure the performance of a poker-playing program both empirically and experimentally. This chapter is dedicated to measure experimentally HuBot's performance by playing thousands of hands.

The unit of measurement is the average number of small bets per hand (sb/hand).

After thousands of trials, this value should converge to a number, representing how good our program performs. The variance of one hand is estimated to be around 6 sb/hand, by poker professional and creator of the CPRG, Darse Billings, in [30]. Therefore, the formula to estimate the variance after *N* hands is:

$$\text{variance} = \frac{6}{\sqrt{N}} \tag{14}$$

For example, the variance after playing 10,000 hands is of 0.06 sb/hand, whereas after 40,000 hands the variance decreases to half, 0.03 sb/hand.

HuBot has been tested experimentally, playing in *Poker Academy Pro 2.5* [37], against several other agents, on a full ring table. This poker software makes the assessment of results easier, by displaying comprehensive graphs and analysis of the player statistics.

In this experimental evaluation, three test scenarios will be presented and discussed. In the first, HuBot plays in the ***advanced*** table of Poker Academy, against an older version of himself. In the second experiment, HuBot shows off his superiority in the ***beginners*** table. In the third and last test, two versions of HuBot play in the advanced table: one with and the other without Opponent Modeling, and the results are compared.

In all these three test scenarios, the table stakes are **$1/$2**, with a starting bankroll of $10,000 for every player.

These tables are limited to a maximum of nine players. This is because the Poker Academy software forces that at least one human player must be playing in the table. For this reason, in the tests, a human player with the name "André" appears in the poker table, but is *sitting out*, and will not be dealt cards.

5.1 Scenario One: Advanced table

Here, HuBot plays against an older version of himself in the *advanced* table of the Poker Academy.

This table contains the world's best AI poker agents in Limit Texas Hold'em: Poki and Simbot. **Poki** is an award winner poker-program, created by the CPRG over a period of many years. **Simbot** is a simulation-based agent uses advanced modeling techniques to simulate the future betting rounds and decide on the most profitable action.

Five Poki agents appear with the names Ogo Pogo, *Sanja*, *Hooke*, *Erasmus*, and *Anders*. Two Simbot's are playing with names *Hari* and *Ginger*.

These agents differ from one another, by having a different playing style. For example, while Ogo Pogo is a very *loose aggressive* player, Anders is much more moderate and has a tendency to check-raise.

The current version of HuBot, version 113, is the result of this work and its implementation is described in this document. Version 90 of the HuBot has a primitive post-flop betting strategy, that doesn't include strategies like *bluffing* and *check-raising*, and a few improvements over other parts of the game have been made since then.

Figure 15 shows the bankroll graph for HuBotv113, after about 27,600 hands have been played.

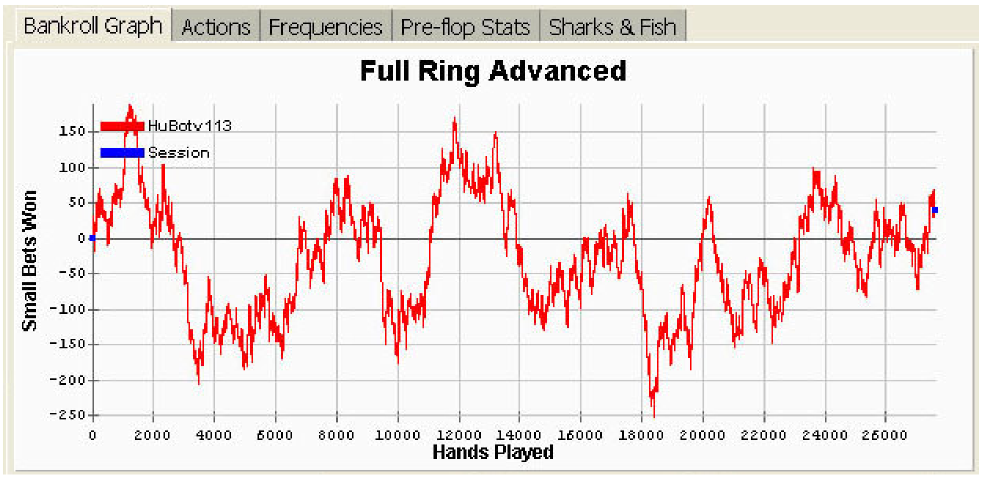


Fig 15: Bankroll graph of HuBot v113, for scenario one.

The income rate of HuBot was 0.00 sb/hand with a variance of about ± 0.035 sb/hand. This value seems to fluctuate a bit in the graph. This can be just due to the stochastic element of poker, or by the Opponent Modeling struggling to track a moving target, as the opponents change their strategies. It's hard to know for sure the reason behind this variation. A better inference could be made by observing the decisions of the players during these hands.

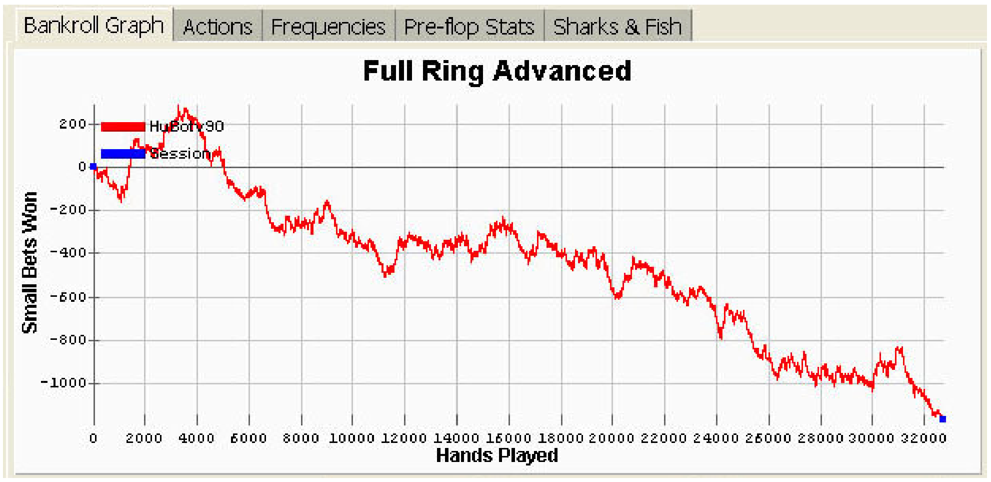
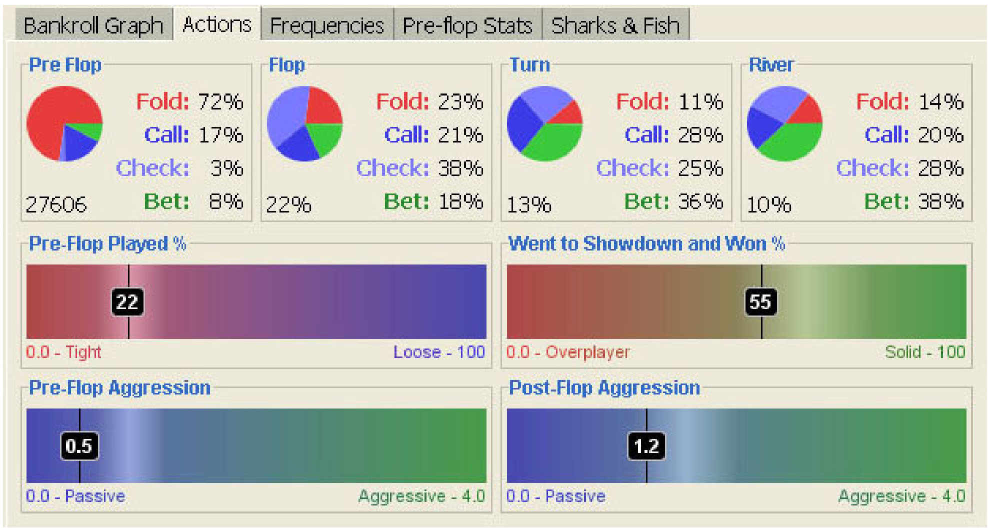


Fig 16: Bankroll graph of HuBot v90, for scenario one.

For HuBot v90, the income rate was -0.04 sb/hand, variance ± 0.03 sb/hand, after 37,000 hands had been played. This graph seems to be very stable, with a continuously decreasing value. In fact, it is safe to say that HuBot v90's opponents have found a way to exploit him. This agent plays a very regular post-flop strategy, without many deceptive decisions. By doing so, his opponent's are able to make a better guess about the cards he holds. Over time, this translates into a slow, but steady loss.

Another interesting statistic concerns the frequencies of actions taken by HuBot v113. This is shown in Figure 17.



Pre Flop
Fold: 72%
Call: 17%
Check: 3%
27606 Bet: 8%

Flop
Fold: 23%
Call: 21%
Check: 38%
22% Bet: 18%

Turn
Fold: 11%
Call: 28%
Check: 25%
13% Bet: 36%

River
Fold: 14%
Call: 20%
Check: 28%
10% Bet: 38%

Pre-Flop Played %
22
0.0 - Tight Loose - 100

Went to Showdown and Won %
55
0.0 - Overplayer Solid - 100

Pre-Flop Aggression
0.5
0.0 - Passive Aggressive - 4.0

Post-Flop Aggression
1.2
0.0 - Passive Aggressive - 4.0

Fig 17: Action frequencies of HuBot v113, for scenario one.

As it can be seen, the pre-flop percentage of hands played is 22% and matches the percentage suggested by Poker Academy. The probability of winning at showdown seems good as well, with more than half times HuBot taking the pot.

This agent is not so aggressive in the pre-flop stage, because it has been adapted to play against other poker agents. And usually, poker-programs tend to be *tight*, and do not *bluff* so often, having strong mathematical models. Therefore, HuBot has also a conservative style in pre-flop, to account for this.

This test shows two positive results:

1. Firstly, that the improvements done to the agent over time increased his performance and that this can be shown experimentally.
2. Secondly, it is very satisfactory to see that the best poker-programs were unable to exploit HuBot v113, as he broke even. By managing complex strategies such as *bluffing* and *check-raising*, it seems the agent made himself a difficult target to be modeled by other agents.

5.2 Scenario Two: Beginners table

Here, HuBot (version 113) faced seven different Jagbots and one Poki, in the *beginners* table. **Jagbots** are simple basic-strategy agents that are unable to adapt to the opponents. The only Poki agent in the table is named *Kenny*, and is a *tight player* with a deceptive style.

After 52,629 hands, HuBot was the player with the biggest *stack*, by a large margin, as seen in Figure 18.

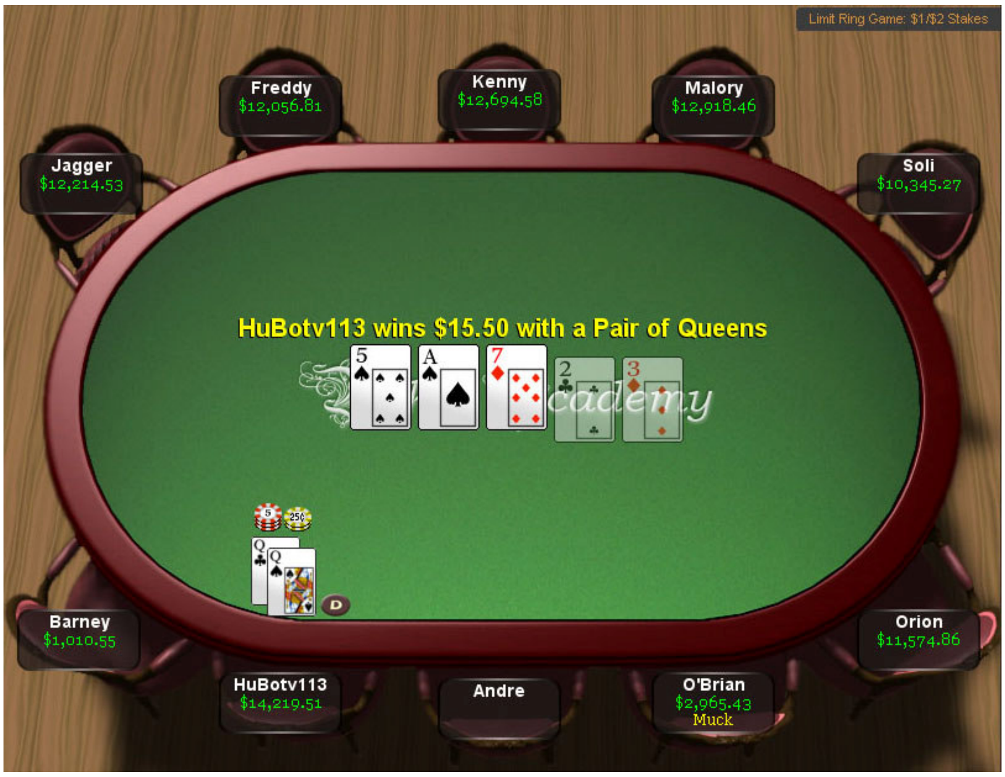


Fig 18: Beginners table, after a HuBot v113's winning.

HuBot's profit rate during the game was very stable at +0.08 bets/hand, variance ± 0.025 sb/hand. In fact, the agent was dominating all the other players, having positive winning rates against each of his opponents, individually.

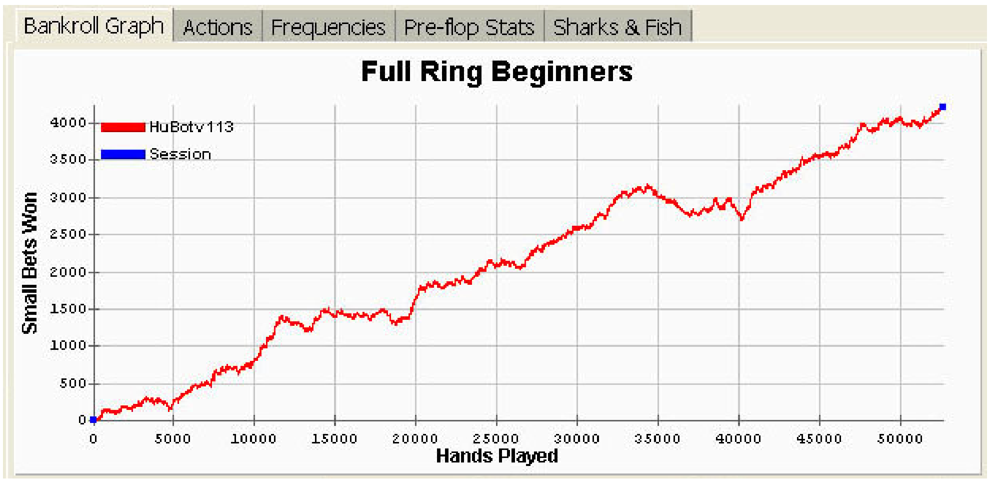


Fig 19: Bankroll graph of HuBot v113, for scenario two.

Another interesting result, is that the agent performed much better against the opponents immediately on his right, than against the opponents on his left. This is because poker is played in a clock-wise order, thus the players on our left usually act after us, gaining a positional advantage. Playing a hand in a *late position* tends to be more profitable.

As it can be concluded from this test, HuBot is capable of exploiting weaker opponents, even with a strong agent in the table. In fact, HuBot was able to outperform *Kenny*, showing a very stable, winning playing style.

It is very important that a poker agent must be able to adapt to the weaker players efficiently, in order to establish his superiority. HuBot proved his ability to do so, by showing a good winning rate progression over more than 50,000 hands.

5.3 Scenario Three: No opponent modeling

This test takes place in the *advanced* table, and its goal is to demonstrate the significance of using Opponent Modeling.

The current agent, HuBot version 113, played against a version of himself that uses no Opponent Modeling, that we'll call HuBot version 113**b**. So, no *weight tables* are kept for the players, and the Hand Strength calculation is done unbiased, assuming an equal distribution over the opponent's possible hands. Also, the *Action Frequencies tables* aren't used, since there is no *re-weighting* done. As a consequence of this, the post-flop *bluff* strategy can't be used, since it is not possible to predict the probability that an opponent will fold when the agent bets.

The *advanced* table has been described in Section 5.1. Basically, it consists of five Poki agents, and two simulation-based Simbots.

By putting these two agents to play in this table, the performance of opponent modeling has been demonstrated without a doubt.

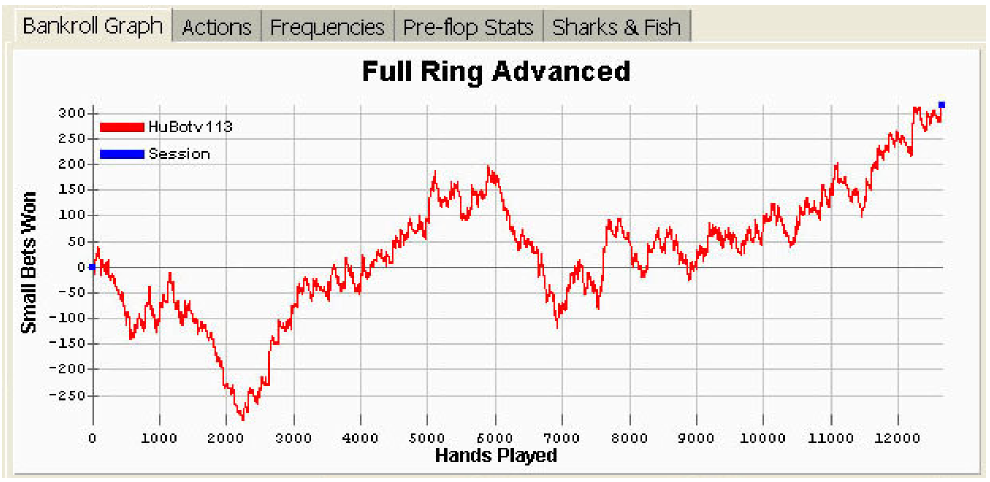


Fig 20: Bankroll graph of HuBot v113 in scenario three.

The test was stopped after 12,642 hands had been played, since the income rate values for the agents could be inferred. In the case of HuBot v113, it was expected that the income rate would be very close to the one obtained in the first test scenario, since they were playing in almost same conditions. This was seen valid, because when the test was stopped the income rate was +0.02 sb/hand, variance ± 0.05 sb/hand.

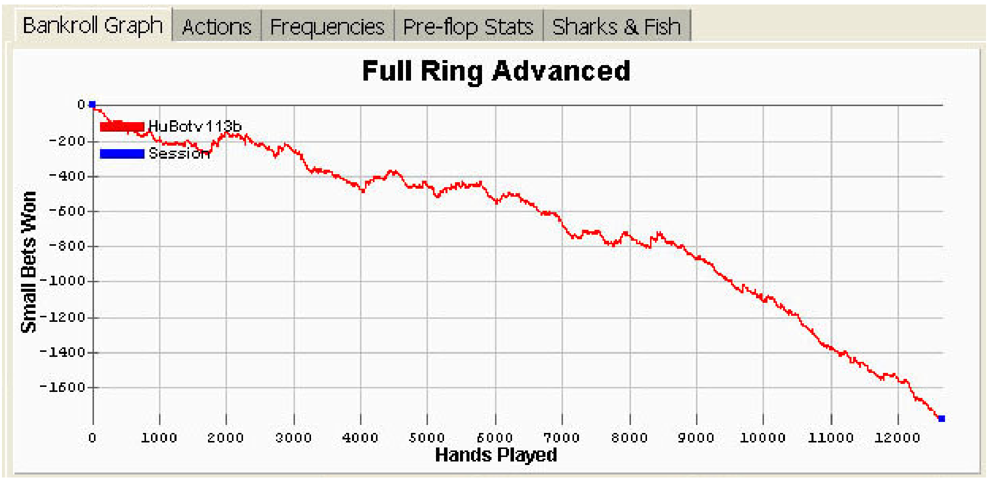


*Fig 21: Bankroll graph of HuBot v113**b**, for scenario three.*

In the case of HuBot v113**b**, the income rate was -0.14 sb/hand, variance ± 0.05 sb/hand, and can be seen by the graph of Figure 21. This result shows the importance of adapting our playing style to the opponents.

The difference between modeling the opponents or not, in this test, was of about 0.16 sb/hand. Although this large number, by itself, doesn't prove that HuBot's Opponent Modeling approach is entirely correct, it establishes the importance of the work done in this component of the project.

Chapter 6

6. Conclusions and Future Work

This work intended to explore the state of the art of computer poker (in Texas Hold'em), and develop a successful intelligent poker agent based on that research.

Texas Hold'em was chosen for its unique properties that separate it from other games, described in the beginning of chapter 2. The domain was then restricted to Limit Texas Hold'em for simplicity, and to be able to focus more in-depth in one variant.

In chapter 3, the rules of the game have been explained in detail, as well as the importance of the position of the players and the size of the table in poker. The difficulties of creating a non-human poker player come from the complexity of this game. In order to surpass all human players, a poker-program must account for the *unpredictability* and *deceitfulness* of human players.

The first goal of this work was to research the most successful approaches used in the past to create an intelligent poker agent. Many papers and thesis have been written about the subject, and AI research groups have been looking into this topic for more than a decade. As a result, many different intelligent agents have been developed.

Approaches have evolved, from a simple case-by-case rule system, to an innovative game-theoretical implementation.

The second goal was to define an architecture based on one successful approach studied before. The approach chosen was a probabilistic formula-based system, as it had proven to shown the best results for the Limit variant of Texas Hold'em, played in a full ring of players.

For the implementation, it was decided to develop the agent under the Meerkat API of Poker Academy, in order to minimize the time spent with the data structures and methods of the state of the game. This time was devoted to develop the intelligence of the agent.

The main goal of this work was to develop the agent. The main components of the agent are the pre-flop betting strategy, the post-flop betting strategy, and the opponent modeling. HuBot relies on an expert formula-based system for the betting strategies. This system was consistently adjusted by an expert poker player, throughout HuBot's versions.

The agent is capable of using complex strategies such as *check-raise* and *bluff*, to avoid having his playing style accurately modelled by his opponents.

A crucial component of HuBot is the opponent modeling. It uses a new re-weighting system, based on the frequency of actions of the opponents. Distinct situations of the game are grouped together in order to be able to rapidly adapt to an opponent's playing style. This type of opponent modeling was proven to increase substantially the performance of the agent, in chapter 5. Further assessments were made in order to test the agent against different opponents, in a variety of scenarios.

The first scenario showed HuBot being able to stand up against the strongest known computer programs for multi-player Limit Texas Hold'em. Against average players, HuBot was able to consistently show profit.

While these results are very promising, extensive tests must be done against human players, to effectively assess the agent's performance, and discover his weaknesses.

Throughout this work, many possible improvements were left undone, due to lack of time. However, if HuBot will be revisited in the future, some topics can be explored in order to increase his performance:

- The use of an Artificial Neural Network, to predict opponent's decisions in real time, as suggested in Section 3.2.2.
- Instead of using only one model of predicting an action (such as statistical table, neural network, etc...), use a *meta-predictor* that calls the other models and evaluates the better decision based on the past accuracy of each of the models. This way, the agent can rapidly change his playing style, if another prediction model is getting a better accuracy in the recent hands.
- The use of *showdown* information (hands shown by the end of the game) is crucial for human players to analyse their opponents' playing style, and has been ignored in computer poker. If explored correctly in opponent modeling, this topic might be able to increase substantially the performance.
- Refine the post-flop betting strategy with more rules and complex strategies, or simply replace it by a Simulation approach (and base HuBot's decisions on the expected value of each possible action).
- Give more credit to the position of the players, and the effect of this property on the actions of the players. The agent must be able to separate a context based on

the relative position of his opponents. E.g. most players are more prone to *bluff* if they are in a late position.

- Improve the *pot odds* post-flop betting strategy, by taking into account the number of players in the formula.
- Establish better default frequencies for the *Action Frequencies table* for the new opponents. These default values should be derived experimentally and adjusted to both the first and second *Action Frequencies table*. To get better initial estimations, different default tables might be considered, by observing how the player action frequencies approach a default table. This is based on the assumption that players can be divided in types: *tight, aggressive*, *loose*, *passive*.
- In the pre-flop, instead of using Income Rate tables, evaluate hands by using the Monte Carlo method for calculating pre-flop *equity*. Make this calculation depend on the *weight tables* of every active opponent.

This work is concluded with a feeling of satisfaction over the proposed goals. However, further work must be done in the agent for it to be able to compete with an expert human player.

References

[1] A. Davidson. Opponent modeling in poker. Master's thesis, Department of Computing Science, University of Alberta, 2002.

[2] D. Billings. Computer Poker. Master's thesis, Department of Computing Science, University of Alberta, 1995.

[3] The University of Alberta Computer Poker Research Group webpage. World Wide Web, 2008.
`http://poker.cs.ualberta.ca/`
[consulted on the 7th of July, 2008]

[4] D. Billings. Ph.D. dissertation. Algorithms and Assessment in Computer Poker. Department of Computing Science, University of Alberta, Canada, 2006.

[5] D. Billings, D. Papp, J. Schaeffer, and D. Szafron. Opponent modeling in poker. In *American Association of Artificial Intelligence National Conference, AAAI'98*, pages 493.499, 1998.

[6] A. Davidson, D. Billings, J. Schaeffer, and D. Szafron. Improved opponent modeling in poker. In *International Conference on Artificial Intelligence, ICAI'00*, pages 1467.1473, 2000.

[7] D. Papp. Dealing with imperfect information in poker. Master's thesis, Department of Computing Science, University of Alberta, 1998.

[8] L. Pena. Probabilities and simulations in poker. Master's thesis, Department of Computing Science, University of Alberta, 1999.

[9] M. Johnson. Robust Strategies and Counter-Strategies. Master's thesis, Department of Computing Science, University of Alberta, 2007.

[10] T. Schauenberg. Opponent Modelling and Search in Poker. Master's thesis, Department of Computing Science, University of Alberta, 2006.

[11] D. Félix. Opponent Modelling in Texas Hold'em. Master's thesis, Faculdade de Engenharia, Universidade do Porto, 2008.

[12] D. Billings, A. Davidson, J. Schaeffer, and D. Szafron. The challenge of poker. *Artificial Intelligence*, 134(1.2):201.240, January 2002.

[13] A. Davidson. Using Artificial Neural Networks to Model Opponents in Texas Hold'em. 1999.

[14] D. Billings, N. Burch, A. Davidson, R. Holte, J. Schaeffer, T. Schauenberg, and D. Szafron. Approximating Game-Theoretic Optimal Strategies for Full-scale Poker. In *Eighteenth International Joint Conference on Artificial Intelligence, IJCAI-03*, pages 661.675, 2003.

[15] D. Billings, A. Davidson, T. Schauenberg, N. Burch, M. Bowling, R. Holte, J. Schaeffer, and D. Szafron. Game-Tree Search with Adaptation in Stochastic Imperfect-Information Games. In *Lecture Notes in Computer Science,* pages 21.34, 2006.

[16] Wikipedia. Game theory. Wikipedia: The Free Online Encyclopedia.
`http://en.wikipedia.org/wiki/Game_theory`
[consulted on the 7th of July, 2008]

[17] D. Sklansky and M. Malmuth. 2+2 website and poker discussion forum. WWW, 1998.
`http://www.twoplustwo.com/`
[consulted on the 7th of July, 2008]

[18] C. Smith and M. Zinkevich. The AAAI'07 poker competition webpage. World Wide Web, 2007.
`http://www.cs.ualberta.ca/~pokert/2007/index.php`
[consulted on the 7th of July, 2008]

[19] Wikipedia. Texas Hold'em. Wikipedia: The Free Online Encyclopedia.
`http://en.wikipedia.org/wiki/Texas_hold-em`
[consulted on the 7th of July, 2008]

[20] Wikipedia. Nash Equilibrium. Wikipedia: The Free Online Encyclopedia.
`http://en.wikipedia.org/wiki/Nash_equilibrium`
[consulted on the 7th of July, 2008]

[21] Wikipedia. Monte Carlo method. Wikipedia: The Free Online Encyclopedia.
`http://en.wikipedia.org/wiki/Monte_Carlo_simulation`
[consulted on the 7th of July, 2008]

[22] Wikipedia. Bayes' theorem. Wikipedia: The Free Online Encyclopedia.
`http://en.wikipedia.org/wiki/Bayes_theorem`
[consulted on the 7th of July, 2008]

[23] Wikipedia. Bayes' theorem. Wikipedia: The Free Online Encyclopedia.
`http://en.wikipedia.org/wiki/Bayes_theorem`
[consulted on the 7th of July, 2008]

[24] T. Salonen. The BLUFFBOT webpage. World Wide Web, 2008.
`http://bluffbot.com/`
[consulted on the 7th of July, 2008]

[25] A. Davidson. Poker Bot Artificial Intelligence Resources. World Wide Web, 2008.
`http://spaz.ca/poker/`
[consulted on the 7th of July, 2008]

[26] A New Guide to the Starting Hands in Texas Hold'em Poker. World Wide Web, 2008.

`http://www.cs.cmu.edu/People/mummert/poker/`
[consulted on the 7th of July, 2008]

[27] PokerStove: Poker Software and Analysis. World Wide Web, 2008.
`http://www.pokerstove.com/`
[consulted on the 7th of July, 2008]

[28] E. Peretz. Hold'em Killer blog. World Wide Web, 2008.
`http://www.holdemkiller.blogspot.com`
[consulted on the 7th of July, 2008]

[29] Poker for Programmers blog. World Wide Web, 2008.
`http://pokerforprogrammers.blogspot.com`
[consulted on the 7th of July, 2008]

[30] A. Davidson, T. Schauenberg, D. Billings. Poker Academy Forums. World Wide Web, 2008.
`http://forums.poker-academy.com`
[consulted on the 7th of July, 2008]

[31] Poker Artificial Intelligence Forums. World Wide Web, 2008.
`http://www.pokerai.org/pf3/index.php`
[consulted on the 7th of July, 2008]

[32] A. Phillips. Man vs. Machine. In *ABC News*, 2007.
`http://abcnews.go.com/Technology/story?id=3409525`
[consulted on the 7th of July, 2008]

[33] D. Silverberg. Online Poker. In *Digital Journal*, 2006.
`http://www.digitaljournal.com/article/35746`
[consulted on the 7th of July, 2008]

[34] D. Parlett, J. McLeod. A History of Poker. 2005.
`http://www.pagat.com/vying/pokerhistory.html`
[consulted on the 7th of July, 2008]

[35] J. Krim. Poker's Popularity Proves a Hot Hand for Gaming Industry. In *Washington Post*, 2004.
`http://www.washingtonpost.com/wp-dyn/articles/A64457-2004Sep5.html`
[consulted on the 7th of July, 2008]

[36] Steve Brecher. Hold'Em Showdown. World Wide Web, 2008.
`http://www.stevebrecher.com/Software/software.html`
[consulted on the 7th of July, 2008]

[37] Poker Academy website. World Wide Web, 2008.
`http://www.poker-academy.com/`
[consulted on the 7th of July, 2008]

[38] J. F. Nash. Non-Cooperative Games. In *Annals of Mathematics 54,* pages 286.295, 1951.

[39] S. Russel, P. Norvig. Artificial Intelligence: A modern approach (2nd edition). 2003.

Appendix A – Glossary of Poker Terms

The following list of poker terms has been adapted from the Glossary of Poker Terms found in [4], and also re-used in [11]. For a more accurate and complete definition of poker terms and expressions, please visit the online Wikipedia: `http://en.wikipedia.org/wiki/Poker_jargon`.

- **All-in**. To have one's entire stake committed to the current pot. Action continues toward a side pot, with the all-in player being eligible to win only the main pot.
- **All-in Equity**. The expected value income of a hand assuming the game will proceed to the showdown with no further betting (i.e., a fraction of the current pot, based on all possible future outcomes).
- **Bad Beat.** An unlucky loss. In particular, losing a game where the opponent probably should have folded, but instead got extremely lucky to win.
- **Bet**. To make the first wager of a betting round (compare raise).
- **Bet for Value.** To bet with the expectation of winning if called (compare bluff).
- **Big Bet**. The largest bet size in Limit poker (e.g., \$20 in \$10-\$20 Hold'em).
- **Big Blind.** (sometimes called the Large Blind). A forced bet made before the deal of the cards (e.g., \$10 in \$10-\$20 Hold'em, posted by the second player to the left of the button).
- **Blind**. A forced bet made before the deal of the cards (see small blind and big blind).
- **Bluff**. To play a weak hand as though it were strong, with the expectation of losing if called (see also semi-bluff and pure bluff , compare bet for value).
- **Board** (or Board Cards). The community cards shared by all players.
- **Board Texture**. Classification of the type of board, such as having lots of high cards, or not having many draws (see dry).
- **Button**. The last player to act in each betting round in Texas Hold'em. Also called the dealer button, representing the person who would be the dealer in a home game.
- **Call**. To match the current level of betting. If the current level of betting is zero,

the term check is preferred.

- **Cap**. (a) The maximum number of raises permitted in any single round of betting (typically four in Limit Hold'em, but occasionally unlimited). (b) (vt) To make the last permitted raise in the current betting round (e.g., after a bet, raise, and re-raise, a player caps the betting).
- **Check.** To decline to make the first wager of a betting round (compare call).
- **Check-Raise.** To check on the first action, with the intention of raising in the same betting round after an opponent bets.
- **Community Cards.** The public cards shared by all players.
- **Connectors.** Two cards differing by one in rank, such as 7-6. More likely to make a straight than other combinations.
- **Dominated**. A Hold'em hand that has a greatly reduced chance of winning against another because one or both cards cannot make a useful pair (e.g., KQ is dominated by AK, AQ, AA, KK, and QQ, but not by AJ or JJ).
- **Draw**. A holding with high potential to make a strong hand, such as a straight draw or a flush draw (compare made hand).
- **Draw Potential**. The relative likelihood of a hand improving to be the best if it is currently behind.
- **Drawing Dead**. Playing a draw to a hand that will only lose, such as drawing to a flush when the opponent already holds a full house.
- **Drawing Hand**. A hand that has a good draw (compare made hand).
- **Dry**. Lacking possible draws or betting action, as in a dry board or a dry game.
- **Equity (or Pot Equity**). An estimate of the expected value income from a hand that accounts for future chance outcomes, and may or may not account for the effects of future betting (e.g., all-in equity).
- **Expected Value (EV)** (also called mathematical expectation). The average amount one expects to win in a given game situation, based on the payoffs for each possible random outcome.
- **Flop.** The first three community cards dealt in Hold'em, followed by the second betting round (compare board).
- **Fold**. To discard a hand instead of matching the outstanding bet, thereby losing any chance of winning the pot.
- **Fold Equity.** The equity gained by a player when an opponent folds. In

particular, the positive equity gained despite the fact that the opponent's fold was entirely correct.

- **Forward Blinds**. The logical extension of blinds for heads-up (two-player) games, where the first player posts the small blind and the second player (button) posts the big blind (compare reverse blinds). (Both rules are seen in practice, with various casinos and online card rooms having different policies for multi-player games that have only two active players).
- **Free-Card Danger**. The risk associated with allowing an opponent to improve and win the pot without having to call a bet (in particular, when they would have folded).
- **Free-Card Raise**. To raise on the flop intending to check on the turn.
- **Game.** (a) A competitive activity in which players contend with each other according to a set of rules (in poker, a contest with two or more players). (b) A single instance of such an activity (in poker, from the initial dealing of the cards to the showdown, or until one player wins uncontested).
- **Game Theory**. Among serious poker players, game theory normally pertains to the optimal calling frequency (in response to a possible bluff), or the optimal bluffing frequency. Both depend only on the size of the bet in relation to the size of the pot.
- **Hand.** (a) A player's private cards (e.g., two hole cards in Hold'em). (b) One complete game of poker (see game (b)).
- **Heads-up.** A two-player (head-to-head) poker game.
- **Hole Card.** A private card in poker (Texas Hold'em, Omaha, 7-Stud, etc.).
- **Implied Odds**. (a) The pot odds based on the probable future size of the pot instead of the current size of the pot (positive or negative adjustments). (b) The extra money a strong hand stands to win in future betting rounds (compare reverse implied odds).
- **Kicker.** A side card, often deciding the winner when two hands are otherwise tied (e.g., a player holding Q-J when the board is Q-7-4 has top pair with a Jack kicker).
- **Large Blind** (usually called the Big Blind). A forced bet made before the deal of the cards (e.g., $10 in $10-$20 Hold'em, posted by the second player to the left of the button).
- **Loose Game.** A game having several loose players.

- **Loose Player.** A player who does not fold often (e.g., one who plays most hands at least to the flop in Hold'em).
- **Made Hand**. A hand with a good chance of currently being the best, such as top pair on the flop in Hold'em (compare draw).
- **Mixed Strategy.** Handling a particular type of situation in more than one way, such as to sometimes call, and sometimes raise.
- **Offsuit**. Two cards of different suits (also called unsuited, compare suited).
- **Open-Ended Draw.** A draw to a straight with eight cards to make the straight, such as 6-5 with a board of Q-7-4 in Hold'em.
- **Outs**. Cards that will improve a hand to a probable winner (compare draw).
- **Pocket Pair.** Two cards of the same rank, such as 6-6. More likely to make three of a kind than other combinations (see set).
- **Post-flop**. The actions after the flop in Texas Hold'em, including the turn and river cards interleaved with the three betting rounds, and ending with the showdown.
- **Pot**. The common pool of all collected wagers during a game.
- **Pot Equity** (or simply Equity). An estimate of the expected value income from a hand that accounts for future chance outcomes, and may or may not account for the effects of future betting (e.g., all-in equity).
- **Pot Odds**. The ratio of the size of the pot to the size of the outstanding bet, used to determine if a draw will have a positive expected value.
- **Pre-fop**. The first round of betting in Texas Hold'em before the flop, beginning with the posting of the blinds and the dealing of the private hole cards.
- **Pure bluff** . A bluff with a hand that can only win if the opponent folds (compare semi-bluff).
- **Pure Drawing Hand**. A weak hand that can only win by completing a draw, or by a successful bluff .
- **Raise**. To increase the current level of betting. If the current level of betting is zero, the term bet is preferred.
- **Raising for a Free-card.** To raise on the flop intending to check on the turn.
- **Rake.** A portion of the pot withheld by the casino or host of a poker game, typically a percentage of the pot up to some maximum, such as 5% up to $3.
- **Re-raise.** To increase to the third level of betting after a bet and a raise.

- **Reverse Blinds.** A special rule sometimes used for heads-up (two-player) games, where the second player (button) posts the small blind and the first player posts the big blind (compare forward blinds). (Both rules are seen in practice, with various casinos and online card rooms having different policies for multi-player games that have only two active players).
- **Reverse Implied Odds.** The unaccounted (negative) money a mediocre hand stands to lose in future betting rounds (compare implied odds (b)).
- **River.** The fifth community card dealt in Hold'em, followed by the fourth (and final) betting round.
- **Semi-bluff** . A bluff when there are still cards to be dealt, with a hand that might be the best, or that has a reasonable chance of improving to the best if it is called (compare pure bluff).
- **Second pair.** Matching the second highest community card in Hold'em, such as having 7-6 with a board of Q-7-4.
- **Session.** A series of games, typically lasting several hours in length.
- **Set.** Three of a kind, formed with a pocket pair and one card of matching rank on the board. A very powerful and well-disguised hand (compare trips).
- **Short-handed Game.** A game with less than the full complement of players, such as a Texas Hold'em game with five or fewer players.
- **Showdown.** The revealing of cards at the end of a game to determine the winner.
- **Side pot**. A second pot for the remaining active players after another player is all-in.
- **Slow-play.** To check or call a strong hand as though it were weak, with the intention of raising in a later betting round (compare smooth-call and check-raise).
- **Small Bet**. The smallest bet size in Limit poker (e.g., $10 in $10-$20 Hold'em).
- **Small Blind.** A forced bet made before the deal of the cards (e.g., $5 in $10-$20 Hold'em, posted by the first player to the left of the button).
- **Smooth-call**. To only call a bet instead of raising with a strong hand, for purposes of deception (as in a slow-play).
- **Suited.** Two cards of the same suit, such as both Hearts. More likely to make a flush than other combinations (compare offsuit or unsuited).
- **Table Image**. The general perception other players have of one's play.

- **Table Stakes.** A poker rule allowing a player who cannot match the outstanding bet to go all-in with his remaining money, and proceed to the showdown (also see side pot).
- **Texture of the Board.** Classification of the type of board, such as having lots of high cards, or not having many draws (see dry).
- **Tight Player**. A player who usually folds unless the situation is clearly profitable (e.g., one who folds most hands before the flop in Hold'em).
- **Time Charge.** A fee charged to the players in a poker game by a casino or other host of the game, typically collected once every 30 minutes.
- **Top Pair**. Matching the highest community card in Hold'em, such as having Q-J with a board of Q-7-4.
- **Trap**. To play a strong hand as though it were weak, hoping to lure a weaker hand into betting. Usually a check-raise, or a slow-play.
- **Trips.** Three of a kind, formed with one hole card and two cards of matching rank on the board. A strong hand, but not well-disguised (compare set).
- **Turn**. The fourth community card dealt in Hold'em, followed by the third betting round.
- **Unsuited.** Two cards of different suits (also called offsuit, compare suited).
- **Value Bet**. To bet with the expectation of winning if called (compare bluff).
- **Wild Game.** A game with a lot of raising and re-raising. Also called an action game.