Methodologies and Tools for Creating Competitive Poker Playing Agents

Luís Filipe Guimarães Teófilo

PROGRAMA DOUTORAL EM ENGENHARIA INFORMÁTICA

Supervisor: Luís Paulo Gonçalves dos Reis (Ph.D.)

Co-supervisor: Henrique Daniel de Avelar Lopes Cardoso (Ph.D.)

2016, January

Abstract

Many researchers have devoted their time to develop software agents intended for strategic games. These agents obtained outstanding results in popular games such as Chess in which current agents cannot be consistently beaten by the best human players. However, for stochastic games with incomplete information there are still no optimal solutions, especially for games with large search spaces, where research is limited by current hardware. Poker is a game that is frequently used to measure progress in this domain, given its key features: simplicity; large number of decision points; hidden cards. Major scientific advances have already been achieved: agents are unbeatable in Head's up Limit Poker. However, in more popular Poker variants, agents are still far from perfect. In this thesis Poker is approached in-depth by addressing all necessary aspects to create Poker software agents, both in scientific and engineering terms. First, new tools for creating and testing agents are shown, namely a tool for automatic online playing. Next, advances on abstraction techniques are shown, namely a new no-domain specific method. Finally, techniques to enhance game play and decision making are analysed and compared. This includes agent architectures based on expert knowledge and optimizations in the usage of the current state-of-the-art algorithm for game playing (Counterfactual Regret Minimization). All developed methodologies were validated on simulated games or real games. Simulations show great efficiency improvements on current techniques. In real games the developed agents achieved a good result on the AAAI Annual Computer Poker Competition (2nd place in the Kuhn track) and, for the first time reported, they were also profitable in real money multiplayer online matches, against human players.

Keywords: Poker; Game Theory; Opponent Modelling; Simulation; Bot; General Game Playing; Game Description Languages.

Resumo

Muitos investigadores dedicam tempo a desenvolver agentes destinados a jogos estratégicos. Estes obtiveram excelentes resultados em jogos populares como Xadrez, tendo ultrapassando o desempenho de jogadores humanos. No entanto, nos jogos estocásticos de informação incompleta não existem soluções ideais, especialmente para jogos com grande espaço de pesquisa, devido a limitações de *hardware*. O póquer é atualmente o jogo mais popular para medir os avanços nesta área pois tem regras simples, elevado número de pontos de decisão e cartas escondidas. Avanços científicos relevantes foram alcançados onde, inclusivamente, foram criados agentes imbatíveis na variante *Head's up Limit*. No entanto, em variantes mais populares, os agentes ainda não são perfeitos. Nesta tese são abordados todos os aspetos essenciais para a criação de agentes póquer, tanto em termos científicos como da engenharia da solução. Primeiro, foram criadas novas ferramentas para criar e testar agentes, com destaque para um programa que permite aos agentes jogarem *online*. De seguida, são abordadas técnicas de abstração, incluindo um novo método independente do domínio do jogo. Por fim, demonstram-se técnicas para melhorar a tomada de decisão, baseadas em arquiteturas de agentes com base no conhecimento de especialistas e otimizações no uso do algoritmo *Counterfactual Regret Minimization*, abordagem com melhores resultados teóricos nesta área atualmente. As metodologias desenvolvidas foram validadas via simulação e jogos reais. Nas simulações foi possível observar melhoramento da velocidade dos algoritmos. Nos testes em ambiente real, os agentes obtiveram bons resultados na competição do AAAI (2º lugar em *Kuhn*) e nos jogos *online*, demonstrando-se que um agente pode ser rentável a jogar contra humanos.

Palavras-chave: Póquer; Teoria de Jogos; Modelação de Oponentes; Simulação; Agentes Automáticos*;* Jogos Genéricos; Linguagens de Descrição de Jogos.

Acknowledgments

I would like to express my appreciation towards my supervisors: Luís Paulo Reis and Henrique Lopes Cardoso, for all the support over the last years: Luís gave me the opportunity to extend my previous Master Thesis work as a Ph.D., again under his supervision, and helped me to get funding for this project; Henrique always gave me the best advice and very good revisions on my work.

I would like to thank to Professor Eugénio Oliveira for giving me good advice on scientific work and for helping me to get work resources (equipment, conferences).

I also give my thanks to the students that finished their master theses under my supervision. The contribution of their work for this thesis was crucial.

On a more personal note, I would like to thank to my beloved girlfriend Rita. She was very important to cheer me up in a difficult phase of my life, where I had to reconcile my job with the writing of this thesis and a house moving.

Finally, and most importantly, I would like to thank to my family members, in particular to my parents and sister. They were and they will always be there for me. They have encouraged me in pursuing Ph.D. studies and they helped become who I am. They have also always supported me on the most difficult times of my life.

This work was supported by the Portuguese Foundation for Science and Technology (FCT), under Doctoral Grant with reference SFRH/BD/71598/2010.

Luís Filipe Guimarães Teófilo

“Where there's a will there's a way”

Samuel Smiles

Contents

List of Figures

List of Tables

Abbreviations and Acronyms

AAAI – Association for the Advancement of Artificial Intelligence.

ACPC – Annual Computer Poker Competition (organized in the AAAI conference).

AI – Artificial Intelligence.

CIG – Complete Information Games

CFR – Counterfactual Regret Minimization.

CPRG – Computer Poker Research Group (University Alberta, Canada).

DEI – Informatics Engineering Department (FEUP).

EGT – Evolutionary game theory (EGT)

FAI – Fake Artificial Intelligence

FCT – Foundation for Science and Technology

FEUP – Faculty of Engineering, University of Porto

GDL – Game Description Language

IIG – Incomplete Information Games

IRC – Internet Relay Chat

LIACC – Artificial Intelligence and Computer Science Laboratory

NE – Nash-Equilibrium

UP – University of Porto

Chapter 1

Introduction

This chapter provides an overview of this thesis and all the developed work that led to its writing. First, the context and motivation of the work are presented in order to justify it. To further emphasize the work's motivation, the main scientific challenges of this kind of work are also listed. Next, the main objectives and contributions of this thesis are described. The chapter is finalized by outlining the structure of this thesis.

1.1 Context

Artificial intelligence (AI) research is a field of study aimed at developing pieces of software and/or hardware that can replace or assist human beings in performing tasks that require intelligence i.e. tasks that are not methodical and that require expert knowledge about on how to deal with unforeseen events. This is contrary to common machines (such as appliances[1]) or software (such as Notepad) whose aim is the systematic fulfilment of tasks that are composed by sets of instructions. Software applications or machines with intelligence also have the capacity to make decisions and control other systematic systems.

In the beginning, the main goal of AI research was to create a Strong AI: an intelligence that imitates brain functions or human behaviour, with the goal of creating intellects that match or exceed humans' one – towards a technological singularity[2] [1]. As years and research went by it was verified that such project was impracticable at

[1] Some modern appliances can perform intelligent tasks such as heat controlling.
[2] A time when machines are so smart that they are able to create smarter versions of themselves.

the time, so AI researchers focused on solving particular problems, with that being known as weak AI. However this paradigm might change in a recent future, as the capacity of CPUs progresses at geometric rate [2]. Projects, such as the Blue Brain[3] – which is an attempt to replicate a synthetic brain by reverse-engineering the mammalian brain down to the molecular level [3] – clearly support the research on strong AIs. However, weak AI research has achieved far greater results than strong AI research.

One of the most explored fields within weak AI research is the development of game playing software agents[4] – autonomous software entities that perform intelligent tasks without human intervention. It is important to differentiate game playing agents from some types of "intelligent" game strategies. In most games with AIs, the software responsible for playing as the computer has access to more information than the human, can control human avatars or control the seeds of the supposedly random events – this AIs are known as Fake AIs (FAIs). Good examples of FAIs can be found in several Texas Hold'em videogames – the FAIs know which cards their human opponents have and they shuffle the cards in a controlled manner to benefit themselves, especially in harder difficulties. This thesis focuses on weak AI for games, but only on game playing agents rather than fake AIs, since despite the great utility of fake AIs, they do not pose any challenge to science and thus not allowing the achievement of this research's goal – the primary goal of researching games is to transport the acquired knowledge to solve other, potentially and arguably more beneficial, real-life problems.

1.2 Motivation

There are many games that pose interesting challenges for AI. Classic games such as chess or checkers serve as a test bed to test AI problems in well-defined domains, with the goal to adapt the developed expertise to real-life problems. Significant results were achieved when developing game playing agents – the most well-known example is the Deep Blue computer, which was the first machine to ever defeat a human chess

[3] Project website: http://bluebrain.epfl.ch/
[4] Software agents are also known as the robots

world champion [4]. Nowadays, current chess programs have completely surpassed human players thus diminishing down the scientific challenge in this kind of games – complete information games (CIG). Moreover, solving games like chess or checkers also greatly differs from solving real-file problems, due to the lack of incomplete information and stochasticity. As John von Neumann says:

> *Real life is not like that. Real life consists of bluffing, of little tactics of deception, of asking yourself what is the other man going to think I mean to do. And that is what games are about in my theory.*
>
> ***John von Neumann***[5]

Therefore, incomplete information games (IIG) provide a much richer domain to study AI and adapt the developed methodologies to other domains. One great example of such games is Poker. Poker is probably the most popular card betting game in the world. It is played by millions around the world and has become a very profitable industry[6], with massive media coverage[7]. Given its popularity and the amounts of money involved (in the order of billions of dollars each year), Poker also became a research subject in other domains such as Economics (the economic impact of gambling [5]) or Phycology (studies of addictive behaviour [6], [7]).

Poker's key features such as incomplete knowledge, risk management, need for opponent modelling and dealing with unreliable information, turned this game into an important topic in Computer Science, especially for AI. These features make it possible to use this game as an easy tool to measure progress in AI research itself. This is so given the fact that in order to assess new approaches one only has to test them against the former ones – these tests can be easily performed by running simulations (the easiness of simulation is actually one of the main advantages of using games).

Dedicated Poker research started about 20 years ago [8]. Even before that, the well-known scientist John Forbes Nash, also used Poker games to demonstrate his also well-known concept of Nash-Equilibrium (NE) [9], which granted him a Nobel Prize and is nowadays a fundamental concept in the Game Theory. Poker presents a radically

[5] In this statement John von Neumann was referring to Chess.
[6] Growth of Poker industry in the news: http://www.newsweek.com/going-all-online-poker-117991
[7] One good example is a TV channel exclusively dedicated to Poker: **The Poker Channel**

different challenge as compared to other games like chess. In chess or checkers, the two players are always aware of the full state of the game. This means that it is possible, in principle, to understand the opponent's strategy just by observing the movement of the game pieces. On the other hand, a Poker game state is partially hidden: each player can only see his/her cards and the community cards (see Chapter 2). Only at the end of each game the opponents *might* show their cards (this may not even happen in Poker!), being for that reason much more difficult to understand and learn how the opponent plays. Poker is also a stochastic game: it admits the element of chance, given that the player cards are randomly dealt, which means that the decisions must be made thinking in probabilities and mathematical expectation over a series of games and not just a particular game (e.g. winning a particular game of Poker does not have the same meaning as winning a chess game because, due to random events, only several games may assess the player's skill level).

1.3 Main Challenges

Several scientific challenges arise while developing contributions to the Computer Poker research domain. While some of these challenges are also present in other games, some are Poker-specific. The main challenges are:

- **Incomplete Information** – there are always hidden cards in a game of Poker. This means that the game state is not fully visible in any stage of the game, thus from a player's point of view there is no fixed solution for a given problem. Therefore the solutions must be probabilistic (e.g. 55% of the times the agent takes option A and in the other 45% the agent takes option B). In sum, the main challenge that arises from incomplete information is the selection of actions and the understanding of the opponents' behaviour (since he or she only picks an action, the agent does not know the probability of that action).
- **Hand Evaluation** – in Poker, players have hands of cards that constitute the game's score. It is difficult to measure how valuable a hand is, since it greatly depends on the opponent: even a hand with low hand strength

(see Chapter 2) can be very powerful against a weak opponent by, for instance, bluffing him or her. Moreover, there are usually[8] a large number of hand combinations (e.g. 2,598,960 in 5-card Poker and 133,784,560 in 7-card Poker).

— **Size of the decision tree** – the Poker game decision tree is usually very large. While there are smaller versions[9] of Poker like 4 card Kuhn Poker with 12 different decisions points (possible sequence of game events), the simplest version of Limit Texas Hold'em has 3.19×10^{14} decisions points (considering card isomorphism[10]) and the simplest version of No-Limit Texas Hold'em has 9.37×10^{71} decision points which would require about 1.241×10^{49} yottabytes of memory to store a full strategy [10]. For this kind of game tree it is absolutely necessary to use the concept of abstraction – grouping similar decision points.

— **Low number of observations** – this is important especially against human players. Since in Poker players usually bet their own money, they have limited resources. Therefore they have short time to understand how his / her opponents are playing before running out of cash. The size of the game discussed above, makes it even harder: the probability of reaching the exact same decision point is very low.

— **Partial Results** – in order to understand the opponent's behaviour and its driving strategy, it is crucial to know which cards he or she had. However, most Poker games do not reach the so-called showdown phase and even when they do it is very usual that some participants are missing because they already forfeit. Even in a showdown round, there is the possibility to muck the cards (see Appendix B for further understanding).

— **No guaranteed optimal solution –** this happens in multiplayer Poker variants. Even a Nash-Equilibrium (NE) strategy profile does not

[8] There are Poker variants that only use a small portion of the deck instead of the full-deck.
[9] These smaller versions are mainly used for research purposes.
[10] Hand isomorphism – There are different hands with exactly the same value (due to the existence of card suits).

guarantee profit, because if there is any kind of coordination between the opponents, an agent with a NE profile may lose. Moreover, despite the robustness of NE strategies, they are not optimized against particular players.

- **Continuous values in bets / translation –** in No-Limit versions of Poker, one of the main challenges in abstraction is to select the correct amount of chips to bet. This is the reason why decision techniques in No-Limit Poker produce much larger trees.

1.4 Main Goals

Taking into account the previously presented context, the aim of this thesis is to contribute with new methodologies and technologies for the development of Poker software agents. Specifically, this research targets the following:

- Explore how methodologies used on the Computer Poker domain can potentially be used or at least hint to the solution of other AI-related problems;
- Create domain validation methodologies and tools for better assessment of scientific advances;
- Present necessary engineering aspects for the construction of Poker agents as opposed to a more theoretical approach;
- Improve the efficiency of current techniques in order to reduce the huge amount of resources that they need;
- Find out how to combine current techniques and technologies to create a Poker agent that finally surpasses human players by being profitable in online multiplayer matches;
- Overcome the limitations of current methodologies on multiplayer games.

Considering the identified goals, some research questions were framed:

- Is it possible to improve current simulation tools for Poker games? If so, will this improvement help on the construction of more competitive Poker playing agents?
- With currently available technology is it already possible for a Poker playing software agent to be profitable in online multiplayer matches with real money bets? If not, what needs to be improved in software agents to do so?
- In which way can abstraction techniques be improved in order to be domain-free and to better represent their corresponding unabstracted games?
- How is it possible to reduce the large number of resources needed by current techniques without compromising the final results?

1.5 Document Structure

The rest of this thesis can be divided into three different groups: Chapters 2 and 3 are the domain presentation chapters; Chapters 4, 5, 6 and 7 describe the contributions of this thesis; Chapter 8 concludes this document by summarizing this thesis's key contributions and by indicating pointers for future research questions. Each of the remaining chapters specifically presents:

- **Chapter 2** presents fundamental background material to understand the contents of this thesis. This includes an overview about game theory and specific concepts about Poker games. Additionally, the main concepts are also formalized for the better description of the contributions of this thesis through the development Chapters.
- **Chapter 3** presents a literature Review for the state of the art material on the Computer Poker domain. This includes a review of simulation systems, bot applications and a summary of approaches for the creation of game playing agents.

— **Chapter 4** presents the main contributions of this thesis in the domain of simulation, by presenting a new simulation system especially developed for researchers. The architecture of the game playing bot software is also presented as well as the Poker variant specification language.

— **Chapter 5** presents the contributions in the domain of game abstraction, by presenting some techniques whose aim is to reduce large games' sizes in order to make those games tractable by game playing algorithms. Most of the presented techniques are Poker specific, but one new domain independent is also presented.

— **Chapter 6** presents the contributions in the domain of game playing. This includes optimizations in current game playing methodologies, opponent modelling techniques, strategy inferring from data and software agent architectures. The validation of each contribution is included in this chapter.

— **Chapter 7** validates the developed agent architectures by presenting practical results of game playing agents' performance in the AAAI scientific competition and in online games.

— **Chapter 8** finalizes this document by providing this thesis's conclusions and giving suggestions on future research work.

Chapter 2

Background

This chapter provides an overview of fundamental notions to understand the main concepts of this thesis – game theory and Poker games. The formalization of certain definitions (used throughout the remainder of the document) is also included in this chapter.

2.1 Importance of Games

Research on strategic games was one of the first sub-domains to be studied in artificial intelligence. At first, this research focused on the development of software agents whose goal was just to win. Later, with the development of Game Theory another goal emerged: maximize utility. This is rather different than wining or maximizing profit as this definition considers wishes, desires, beliefs or even emotions of the agents [11].

Utility is the measurement of the agent's satisfaction towards the completion of its goals. In the case of games it is usually associated with the game's payoff, but not necessarily. One good example can be found in [12], where an agent theoretically mimics human behaviour: it loses on purpose against a specific opponent for whom it nurses sympathy. From the utility we can take another important measure of extreme importance to game players – the mathematical expectation of an action. This gives us how much we get on average from an action by providing its average utility.

Many known scientists dedicated their time to develop game playing agents with intelligent strategies for games like chess because those games have a clearly defined set of rules and goals which allows them to be an easy domain to experiment on [8] – the main benefit resides on the fact that it is relatively easy to validate new approaches, especially through simulation of game plays. This presents an advantage as it a form of low cost validation of new AI approaches as it is possible to accurately measure the degree of success of a particular approach just by comparing results of many games played against programs based on other approaches or human players – the results are usually measured in terms of agent's average utility growth. This means that games have a well-defined metric for measuring the development progress – in other words, it is possible to determine with more certainty if one is approaching (or not) a more optimal solution to solve a given problem.

Another advantage of games research is that the knowledge gained from solving them can and has been used to solve other AI-related problems.

These and other features like the fact of games having a "recreational factor" and a great impact in the entertainment industry today, make games an extremely important challenge for AI.

2.2 Game Theory

Game Theory is a branch of applied mathematics that models strategic situations – games – in which one's success depends on one's opponents choices [13]. This field of research was initially introduced by John von Neumann in 1928 [14].

Game Theory models problems or situations as a game. A game is any strategic situation that can be described by the following features (for more details on these features see Section 2.3):

- Set of players or actors (at least 2);
- Set of possible moves (decisions) for each player;
- Set of strategies;
- Set of rules;

- State;
- Utility (game's payoff).

The game players are the entities that participate in the situation. Each one can follow its own strategy in a competitive environment with the objective of maximizing the utility (score obtained by the player at the end of the game). For instance, in a game like Poker, the utility would be the money balance of the player at the end of the game. The winning player(s) always have a positive balance, getting for that reason a positive utility. In contrast, the losing players get a negative utility. A game also has a state that represents the current value of the variables involved in the game. Any game move made by any player may alter the game state and the current state may terminate the game, depending on the game's rules.

2.2.1 Strategies in games

In a game, the agents' behaviour is described by their strategies – a function that receives as parameter a game state or information set – in the case of incomplete information games – and returns an action (or decision). Commonly, in game theory, three types of game strategies are considered:

- **Pure Strategy**: for each specific game state, the players always make the same move;
- **Mixed Strategy**: the player assigns a probability to each pure strategy and stochastically picks one;
- **Totally Mixed Strategy**: a particular type of mixed strategy. It consists of a set of pure strategies where each has a strictly positive probability of being chosen.

It should be noted that in game theory each player has its own strategy set which is smaller than the set of all possible strategies for the game in question.

2.2.2 Extensive-form games

An extensive-form game is a representation of a game in game theory in a form of a game tree. This representation allows for the demonstration of certain games'

important aspects, like the sequencing of players' possible moves or their possible choices at every decision point.

One example of such representation of a simple game (Rock Paper Scissors) can be seen in Figure 1. The numbers on the bottom of the game tree represent the game's payoff for each player. The letters R, P and S represent the possible game's actions, respectively actions Rock, Paper and Scissors.

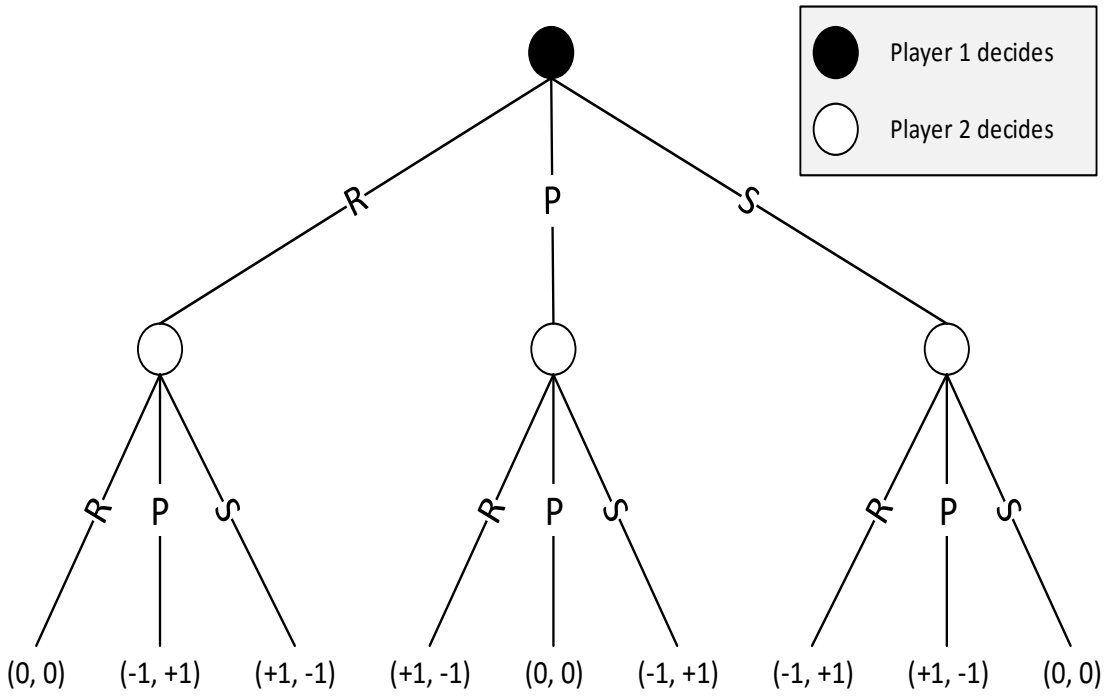


Figure 1 – Rock Paper Scissors represented in extensive-form

Table 1 – Rock Paper Scissors represented in normal-form

		Player 2		
		Rock	**Paper**	**Scissors**
Player 1	**Rock**	(0, 0)	(-1, +1)	(+1, -1)
	Paper	(+1, -1)	(0, 0)	(-1, +1)
	Scissors	(-1, +1)	(+1, -1)	(0, 0)

2.2.3 Normal-form game representation

The normal-form game is a matrix representation of the game, which shows the players, strategies and the final payoffs. One example of that representation of a simple game (Rock Paper Scissors) can be seen in Table 1. Each matrix position contains the payoffs for both players considering the chosen actions.

2.2.4 Nash-equilibrium in Game Theory

John Forbes Nash is a mathematician who did remarkable research in the area of game theory. He introduced the famous Nash-Equilibrium-Theory [9] in 1950. The Nash-Equilibrium represents a set of mixed strategies in a game where if any player ***i*** were to change its strategy, it would decrease its overall utility. This change is sole, i.e. it assumes that every other player maintains its strategy.

A set of Nash-equilibrium strategies usually provides a very stable solution, especially for developing game playing agents for 2 player games. Usually, a Nash-Equilibrium is computed by self-play[11] algorithms or linear programming which results in two strategies (one for player 1 and the other for player 2). Then, the idea is for the game playing agent to use player 1's strategy when it is in the player 1 position, otherwise it uses the strategy of player 2. This way, an agent can assure a minimum average utility against any opponent. The biggest disadvantage of this method is that it is conservative – it does not optimize results against particular players. When the game is unsolvable, an approach of this type may even not guarantee positive utility – it only warrants that the utility will not go below or above a certain threshold. One example of such a game is Kuhn Poker for 3 players (see 2.5.3). Nash-Equilibrium can be formally defined as follows:

$$\forall i, x_i \in S_i, x_i \neq x_i^*: f_i(x_i^*, x_{-i}^*) \geq f_i(x_i, x_{-i}^*)$$

EQ1

[11] It is a type of reinforcement learning algorithm for games. It consists of an approach where the game playing agent optimizes its strategy over time by playing several games against itself.

The notation for EQ1 is defined below:

(S, f) is a game with n players, where:

- i player
- x_i a mixed strategy profile[12] for player *i*.
- x_i^* a mixed strategy profile for player *i* that is in Nash-Equilibrium.
- S_i the set of all possible strategy profiles for player *i*.
- *-i* an opponent
- f_i a function that takes two strategy profiles as arguments and returns the average utility for player *i* in matches between those strategy profiles.

The equation formalizes that if player *i* deviates from his * profile (Nash-equilibrium) strategy, its utility will never increase.

2.2.5 Evolutionary Game Theory

Evolutionary game theory (EGT) is a branch of game theory that studies games in a biological context base on Charles Darwin and Herbert Spender principle: *"The Survival of the Fittest"*. Thus, EGT is based on the idea of natural selection: over time the population evolves and during that evolution the stronger individuals will survive and the weaker ones will perish. In other words, the population always evolves in order to adapt to the environment that it is in, therefore optimizing its processes taking into account the available resources.

Evolutionary game theory was first introduced [15] by John Maynard Smith in 1973. He found that the fitness of a strategy should not be measured in isolation; it should be measured in interaction with other strategies. Thus, EGT analyses the payoff of a set of strategies rather than individual payoffs, thus focusing on the dynamics of strategy shifting within a population.

[12] Strategy profile - it is a tuple of probabilities of a given player performing each game's possible action: the sum of the tuple's elements is always 1.

2.3 Formalizing Extensive-form Games

An extensive-form game is a generic representation of a sequential decision problem in form of a tree where each edge represents a decision and each node represents a sequence of performed actions (history). The history is hereinafter denoted by *h* considering that *h* ∈ *H*, being *H* the set of all possible game sequences according to the game's rules. Also consider *h'* a history-prefix where *h* = *h'* || *x*. Therefore, a game *G* can be represented as the following tuple:

$$G = < H, Z, N, A, a, u, p > | Z \subset H$$
$$where$$
$$a: H \rightarrow A' : A' \subseteq A$$
$$u: N \times Z \rightarrow \mathbb{Q}$$
$$p: H \rightarrow N$$

EQ2

Z is a subset of *H* and represents the game's terminal nodes i.e. the nodes where the game ends. *N* represents the set of players in the game and *A* is the set of all possible actions.

An extensive-form game also requires the definition of three functions. Function *a* gives the set of all possible actions for a given node (or history) *A'* (that is a subset of *A*) where for any particular node *z* ∈ *Z* we have that *a*(*z*) = ∅ and for any particular node *h* ∈ *H**Z* we have that *a*(*h*) ≠ ∅. Function *p* returns the acting player of any game sequence. Finally, function *u* returns the utility (or score) of a given player at a terminal node.

2.4 Game Classification

Games can be classified taking into account different aspects such as visibility of the game state, duration of the game and the occurrence of random events. Some possible game classifications will now be presented.

It is possible to classify games taking into account the visibility of the game state:

- **Complete Information Games** – when the state of the game is always available to every player. Most board games are complete information games.
- **Incomplete Information Games** – when the state of the game is hidden or partially hidden. Most card games are incomplete information games.

There are two types of classifications regarding the existence of a playing history:

- **Sequential Games** – when the game has sequences of actions and its players play in turns (not necessarily rotated). In this kind of games, players can take into account past opponents' decisions because their actions are observable.
- **Simultaneous Games** – there is no history, the players decide simultaneously.

There are two types of classifications regarding cooperation between players:

- **Cooperative Games** – the players' utility could be shared, i.e. each player does not depend on its individual success but rather its group's success. Not to be confused with each player's individual notion of utility.
- **Non Cooperative Games** – the utility is individual, i.e. the players are only interested on their own individual success.

Taking into account the duration of the game, it is possibly to classify it as:

- **Finitely Long Game –** game that cannot last forever.
- **Infinitely Long Game –** game that can last forever – they can loop forever between the same states.

There are two types of games regarding the sum of payoffs at the end:

- **Zero-sum Games** – the sum of all players payoff is 0, which means that for a player to win a certain amount of points, one or more players must lose the same amount.
- **Non-zero-sum Games –** the sum of all players payoffs is not 0. One good example is the Casino commissions on Poker or Blackjack games, which turn Zero-sum games into non-zero sum games.

Regarding the occurrence of random events, there are two types of games:

- **Deterministic Games –** no occurrence of random events;
- **Stochastic Games** – occurrence of random events (e.g. dice roll).

2.4.1 Examples of games

In order to better understand the types of games described previously, Table 2 is presented – it contains examples of several well-known games and their classification (Blank – it has the opposite classification, ✓ – it has that classification, ? – it may have both classifications, depending on the version of game's rules).

Table 2 – Examples of games and their respective classifications.

Game	Complete Inf.	Sequential	Cooperative	Finite	Zero-sum	Deterministic
Poker		✓		✓	**?**[13]	
Stock Market		✓	**?**		✓	
Minesweeper		✓		✓		
Backgammon	✓	✓		✓	✓	
Checkers	✓	✓		✓	✓	✓
Go	✓	✓		✓	✓	✓
Chess	✓	✓		✓	✓	✓
Prisoner's dilemma	**?**			✓		✓
Monopoly		✓	✓	✓		
Go-Fish		✓	✓			
Football	✓	✓	✓	✓		✓
Diplomacy			✓		✓	✓

[13] Poker is not zero-sum in casinos.

2.5 Poker

> *"Poker is a generic name for literally hundreds of games, but they all fall within a few interrelated types"* [16].
>
> ***David Sklansky***[14]

Poker is a card game in which players bet that their current hand is stronger than the hands of their opponents. All bets throughout the game are placed in the pot and, at the end of the game, the player with the highest ranked hand wins. Alternatively, it is also possible to win the game by forcing the opponents to fold their hands by making bets that they are not willing to match. For this reason, since the players do not know the cards of the opponents, it is possible to win the game with a hand with lower score, by convincing the opponents that one's hand is the highest ranked.

2.5.1 Poker Game Classification

By using the classifications presented in Section 2.4, most Poker games can be classified as follows:

- **Incomplete Information**: because the player does not know the opponents' cards.
- **Non Cooperative**: in Poker there is no cooperation between players. Each player wants to maximize his/her profit. Also, cooperation in Poker is considered cheating.
- **Finitely Long**: a Poker game cannot last forever, because each player has a finite amount of cash. Even if players were continuously raising, they would eventually run out of money which means that they would be forced to go all-in.
- **Zero-sum**: the money won by any player is lost by others, so the sum of gains and losses is always 0. However, when played in casinos, Poker is no longer a zero sum game because the casino is entitled to a percentage of the pot.

[14] David Sklansky – an well-known and renowned expert in gambling.

— **Stochastic**: the cards are randomly dealt to all players.

— **Sequential**: in Poker, players play sequentially. Typically the history sequences are very large as discussed in [10].

2.5.2 Texas Hold'em Poker

Texas Hold'em is a Poker variation that uses community cards. This variant of Poker was chosen because its rules have specific characteristics that allow new developed methodologies to be adapted to other Poker variations with reduced effort [17].

2.5.2.1 Rules

At the beginning of every game, two cards are dealt to each player. The dealer player is assigned and marked with a dealer button. The dealer position rotates clockwise from game to game. After that, the two players to the left of dealer post the blind bets. The first player is called small blind, and the other one is called big blind. They respectively post half of minimum and the minimum bet. The player that starts the game is the one on the left of the big blind. One example of an initial table configuration is shown in Figure 2. The dealer is the player at seat F and the small and big blind players are respectively the A and B seats.

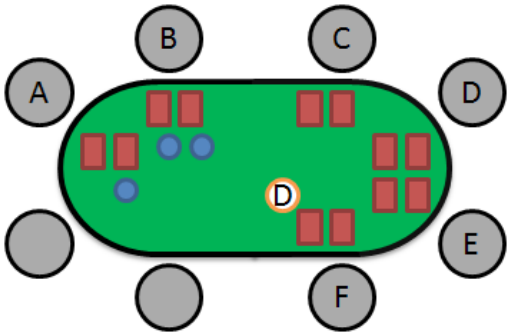


Figure 2 - Poker table layout (from [18]**)**

The first player to act is the player to the left of the big blind (player **C**) and the next player is the closest one to the left of the current player. Each player can choose one of the following actions:

— **Call:** match the current highest bet. If it is not necessary to put more money in the pot, this action is also known as **Check**.

- **Raise:** bet higher than the current highest bet. If the player bets its entire stack, this action is known as **All-In**. If no one has bet/raise previously, the action is called **Bet**.
- **Fold:** forfeit the hand and thus give up the pot. All the money previously put into the pot cannot be recovered by the folding player.

In order to continue to dispute the pot, a player must either call or raise the maximum current bet. In No Limit Texas Hold'em there is no bet limit, therefore the value of the bet can go from the minimum bet up to the player's full bankroll. After all the remaining players either called their hands or went all-in, a round is finished. There are four betting rounds in Texas Hold'em, where in each round new community cards are revealed:

- **Pre-Flop:** no community cards;
- **Flop:** three community cards are revealed;
- **Turn:** the forth community card is revealed;
- **River:** the fifth and final community card is revealed.

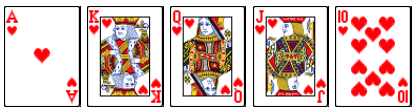
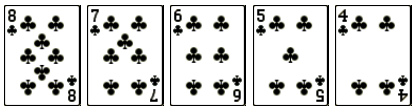
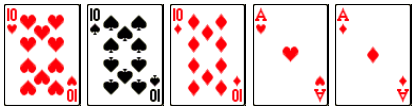
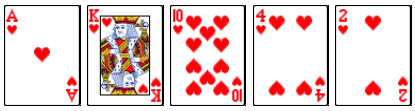
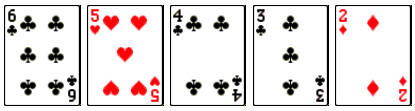
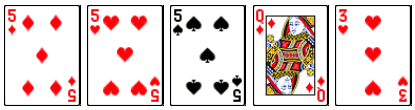
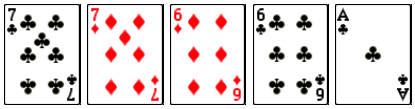
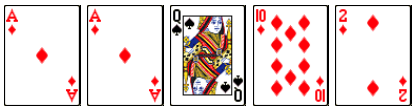
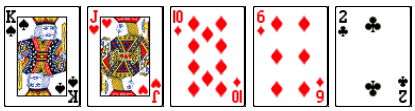
After the river, if at least 2 players agree to call the pot, the showdown round comes. In the showdown players may show their cards and the one with the best hand wins the pot. If two players or more have similar ranked hand, there is a tie and the pot is divided. Next, the hand ranks will be discussed.

2.5.2.2 Hand Ranking

A poker hand is a set of five cards that identifies the score of a player in Poker. The player's hand is made by combining the player's personal cards with the community cards – cards that belong to all players. The set of five cards comprising the pocket and community cards that has the highest possible score is the player's hand ranking.

In Table 3 it is possible to see every possible Poker hand with a description and an example of combination, in descending order of score. In case of draw, for hands of at least one pair, the winning hand is the one with higher ranked cards that are not kickers (see the definition of kicker in Appendix B). If a draw persists, the winner is the one that has kickers with higher ranks.

Table 3 – Poker Hand Ranks

Hand Description	Hand Example
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: Any set with four cards with the same rank.	
Full House: Three cards with the same rank plus two cards with the same rank.	
Flush: Any set with five cards of the same suit, but not in sequence.	
Straight: Five cards in sequence, but with different suit.	
Three of a kind: three cards with the same rank.	
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 with the same rank and three kicker cards.	
High Card: Any hand that does not qualify as one of the better hands above. Ranked by top card, then the second card and so on.	

2.5.3 Other versions of Texas Hold'em Poker

In order to study very large games such as Texas Hold'em Poker, sometimes simplified versions of the game are created mainly for research purposes. These versions do not allow for testing the scalability of the solutions, but they permit observing the algorithms' behaviour.

The most popular simplified versions of Texas Hold'em currently used in academics are **Kuhn** Poker and **Leduc Hold'em** Poker.

Kuhn Poker is the simplest version of Texas Hold'em. It uses a deck only containing 4 cards of the same suit and with different values (e.g. Jack of Spades, Queen of Spades, King of Spades and Ace of Spades) and it is played by up to 4 players. Each player receives a private card from the deck which remains hidden throughout the game to his or her opponents. Players have to put an ante of 1$ and then the game starts. Each player can do one of the following actions:

- **Bet** – put 1$ on the pot.
- **Pass** – don't put any cash in the pot. If any player has betted before, this means that the player that is passing forfeits the game.

To better show the simplicity of this game, Figure 3 illustrates a partial game tree, for one possible configuration of card dealings in Kuhn Poker (player 1 gets a Queen and players 2 gets a King).

For each configuration of card dealings, there are only 4 decision points or information sets (see Section 2.5.4.3) in Kuhn Poker. Since there are only 4 possible cards, this game only has 16 information sets (see Table 4 for the size of simplified Poker games).

Table 4 – The size (number of information sets) of simplified versions of Poker

Number of Players	Kuhn	Leduc Hold'em
2	16	64
3	48	192

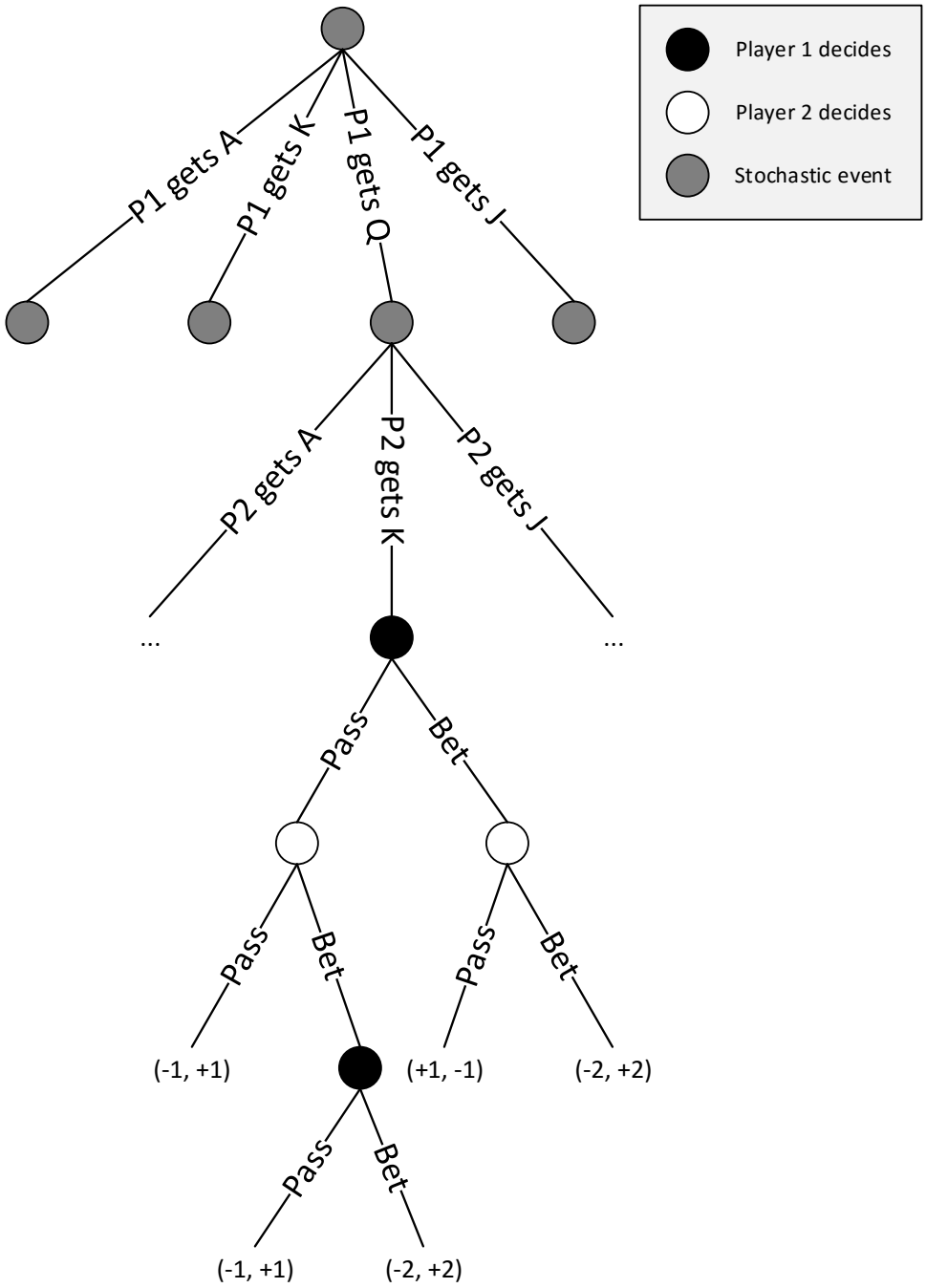


Figure 3 – Partial game tree for Kuhn Poker with 2 players

Leduc Hold'em is a little bit more complex version of Kuhn Poker. The deck has the double number of cards, 4 different values and two suits (e.g. Ace of Spades, Ace of Clubs, King of Spades, King of Clubs, Queen of Spades, Queen of Clubs, Jack of Spades and Jack of Clubs). The main difference is the existence of another betting round, which shows up a community card (which allows players to get the rank pair).

The number of information sets per round is still 4, so there are 16 decision points for each deck card.

2.5.4 Formalizing Texas Hold'em Poker

2.5.4.1 Scoring

At the beginning of a game *G* (see Section 2.3), each player $i \in N$ is given a set of two playing cards (private cards) which we will denote as $\mathrm{P_i} \subset \mathrm{D}$, where *D* is the deck – set of all playing cards (usually a regular 52 card deck without Jokers) – and $\forall i,j \in N$: $P_i \cap P_j = \emptyset$. The private cards P_i are only visible to player *i* and may never be unveiled to other players (only if the game reaches a showdown[15]). At certain moments of the game, some shared cards are revealed – we will denote $S \subset D$ the set of shared cards and $S_r \subseteq S$ the set of visible shared cards at round $\mathrm{r} \in \{\mathrm{preflop}, \mathrm{flop}, \mathrm{turn}, \mathrm{river}\}$, where $\forall \mathrm{i} \in \mathrm{N}: \mathrm{S_r} \cap \mathrm{P_i} = \emptyset$ and $\forall r: S_r \subset D \wedge \mathrm{S_r} \cap P_i = \emptyset$. The shared cards are always visible to all players and are used in combination with the private cards to determine a particular player's *score*. For any No-Limit Poker variant, $S_{preflop} \subset S_{flop} \subset S_{turn} \subset S_{river} = S$ (for Texas Hold'em Poker: $|S_{preflop}| = 0$, $|S_{flop}| = 3$, $|S_{turn}| = 4$, $|S_{river}| = 5$).

In Poker, the score of a player *i* is given by the best possible subset of five cards: $[P_i \cup S]^5$ where the *score* is maximized, being $score : [D]^5 \rightarrow \mathbb{N}^+$ a function that returns the score of a 5 card set. Therefore, for any remaining pair of players *i* and *j*, player *i* wins against player *j* in the conditions of EQ3.

$$max \;\; score([P_i \cup S]^5) \geq max \;\; score\left(\left[P_j \cup S\right]^5\right)$$

EQ3

The score of 5 card sets is divided in ranks (*High Card*, *Pair*, *Two Pairs*, *Three of a Kind*, *Straight*, *Flush*, *Full House*, *Four of a Kind* and *Straight Flush*), each of which is divided into several sub-ranks. The total number of sub-ranks is 7462, therefore $\forall w \in [D]^5: score(w) \in [0;\ 7461]$.

[15] Showdown – a game's terminal node with at least 2 standing players and all bets matched.

2.5.4.2 Rules and Utility

After dealing the cards, the game begins. The game is played in turns that are grouped in four Rounds (Pre-Flop, Flop, Turn and River). In each player's turn, he or she can choose one action which may or may not increase the pot value (prize).

A round ends when all standing players have bet the same amount (but each one must act at least once in that round). When the last round finishes, the player with the highest ranked set of cards wins the game and collects the pot. Alternatively, it is also possible to win the game by inducing opponents to fold by making bets that they are not willing to match. Thus, since players' cards (pocket cards) are hidden, it is possible to win the game with a lower score hand. This particular feature of the game's rules makes it difficult to assess a player's decision. Regardless of the winning situation, the condition on EQ4 must always be verified (definition of zero-sum game).

$$\forall z \in Z: \sum_{i \in N} u(i,z) = 0$$

EQ4

However, usually (but not only) in online Poker the game is not zero-sum due to the casino's profit margin $e \in [0,1]$. Considering $\mathrm{e} \neq 0$, the real utility of player *i* in node *z* is usually given by $u(i,z) \times (1-e)$ if $\mathrm{u(i,z)}$ is positive and $\mathrm{u(i,z)}$ otherwise. In order to complete the definition of a Poker game, we define the new game tuple based on *G* as specified in EQ5.

$$G_P = \left(\begin{array}{c} H, Z, N, \mathrm{A}, \mathrm{P}, \mathrm{S}, a, \mathrm{p}, u, \\ s: N \times H \to \mathbb{Q}_{\geq 0} \\ b: N \times H \to \mathbb{Q}_{\geq 0} \\ r: H \to 2^R, |r| > 0 \\ c: H \to \mathbb{Q}_{\geq 0} \\ v: H \to 2^S, |v| \geq 0 \end{array} \right| Z \subset H$$

EQ5

First, the sets *P* and *S* were included and they respectively correspond to the private and community card sets ($\forall i: P_i \in P$). Functions *s*, *b*, *c*, *v*, and *r* were added to the original definition of *G*. Function *s* denotes the amount of remaining cash and *b* the amount of cash betted by a particular player for a given history *h*, which means that $s(i,h) + b(i,h)$ for any *i* and *h* is the amount of cash of player *i* at the start of the

game. Function *c* returns the value of the current maximum bet. Function *v* returns the visible shared cards for a given history. Finally, *r* is the function that determines the set of remaining players for a given history (it excludes the players that have folded). Given these functions, we can determine the utility of a player. The value of the pot in $h \in H$ is $\sum_i^N b(i,h)$ then, given Texas Hold'em Poker rules, player *i*'s utility in a terminal node *z* is given by EQ6.

$$u(i,z) \in \left\{ -b(i,z), \sum_j^N b(j,z) - b(i,z) \right\} \Big| \, i \in N \wedge z \in Z$$

EQ6

Given these definitions we can also detail the *a* function, which given a history returns the possible betting amounts. The No Limit variant of Texas Hold'em Poker is characterized for having no limits in bets – the players can raise up to their remaining money (see EQ7), where 0 corresponds to a *fold* action, the lower limit $\min\big(s(p(h),h), c(h) - b(p(h),h)\big)$ to a *call* and the higher limit $s(p(h),h)$ to an *all-in*. The lower and the upper limit might be equal, if the player doesn't have enough cash to call – in that case, the player goes all-in.

$$\forall h \in H : a(h) \in \begin{bmatrix} \min\big(s(p(h),h), c(h) - b(p(h),h)\big), \\ s(p(h),h) \end{bmatrix} \cup \{0\} \wedge A = \mathbb{Q}_{\geq 0}$$

EQ7

2.5.4.3 Information sets

An information set is the name of a decision point in Poker; contrarily to complete information games, a player in Poker does not have the full game state information. Poker information sets $I_{i,h} = \{h, P_i, v(h)\} \big| I_{i,h} \in I$ are composed of the game's action sequence, the player's private cards and the visible community cards. Other features can be extrapolated from the basic features, such as the Hand Strength measure, later described in this thesis.

2.6 Summary

In this chapter this thesis's domain was presented by clarifying several basic concepts about game theory and research in games. The game object of this study – Poker – was also introduced by explaining its most important rules and motivation for its research. The main Poker variants used in this thesis were also presented, namely Kuhn, Leduc and Texas Hold'em Poker.

Chapter 3

Literature Review

This chapter describes the current tools, trends, techniques and approaches into building artificial intelligent programs capable of playing Poker. Some brief summaries of areas that may be required for the development of this thesis work are also presented: Section 3.1 gives a brief overview about what is being done in this thesis domain and by whom; Section 3.2 summarizes the current algorithms used by the research community. Section 3.3 presents the main agents that were developed until today. Section 3.4 presents ways on how to efficiently compute ranks of sets of cards. Section 3.5 presents algorithms for computing estimators to aid Poker game playing. Sections 3.6 and 3.7 present expert knowledge. Sections 3.8 and 3.9 present the tools for respectively simulating games and having agent playing online; finally on Section 3.10 a brief summary about emotional agents is given.

3.1 Computer Poker Research

Research on computer Poker has been active over the past 20 years, which is evidenced by the relatively high number of publications in top conferences [19]–[23] and journals [17], [24], as well as master and doctoral theses [25]–[27]. However, none of these works focused on creating Poker players to match with humans. Besides Darse Billings work [25], most of the approaches focused more on the theoretical aspects of game-playing, making them only valid on more theoretical scenarios with theoretical environments. An exception to this is a recent achievement, where a perfect agent was

created for Texas Hold'em Head's On Limit [24]. However, the described approach is (currently) unfeasible for multiplayer no-limit Poker, due to memory and CPU capacity constraints. For this reason there is still a long way to go to create a Poker agent that is capable of consistently beating the best human players, especially in multi-player environments.

3.1.1 Research Groups

The most relevant work in the area was mainly done by research groups exclusively dedicated to Computer Poker. These are the most relevant institutions that produce relevant research work in the area:

- **Computer Poker Research Group (CPRG) at University Alberta**[16] – CPRG is probably the most active group on computer Poker research in the world today. They have produced the first Ph.D. thesis [25] about computer Poker, many other master theses with quite relevant advances and have published several papers in the top artificial intelligence conferences. Most of the research conducted in this group emphasizes on game theory applied to Poker. In most editions they achieve medals in the Annual Computer Poker competition as well (especially on the Limit competitions) [28]. To start exploring this domain, it is very important to thoroughly study the work done by this group.

- **Carnegie Mellon University**[17] – CMU does not have a research group exclusively dedicated to Computer Poker. However, some CMU students, coordinated by Professor Tuomas Sandholm, developed several winning agents at the Annual Computer Poker competition, especially in the No-Limit competition (they developed several abstraction techniques for no-limit Poker).

- **Faculdade de Engenharia da Universidade do Porto / LIACC** – Research in computer Poker started at FEUP around 2008 with two Master theses. Like CMU, there is no research group just for Poker. However, several

[16] CPRG homepage: http://poker.cs.ualberta.ca
[17] See some CMU Poker papers at http://www.cs.cmu.edu/~sandholm/

publications were made during the last years (see Appendix A for examples).

3.1.2 Conferences and competitions

The most relevant work in the domain of Computer Poker can be found in the following conferences. These conferences are top conferences in the area of artificial intelligence, with very low acceptance rates (on average < 10%):

- **AAMAS** – International Conference on Autonomous Agents and Multi Agent Systems
- **AAAI** – International Conference on Artificial Intelligence
- **IJCAI** – International Joint Conference on Artificial Intelligence

There is also relevant work in top journals such as:

- **Artificial Intelligence** Journal
- **The International Computer Games Association** Journal
- **Science**

Most of the work is tested and validated in agent competitions. The most relevant competitions are organized by the CPRG:

- **CPRG Annual Poker Bot Competition**[18] [28] – The Annual Computer Poker Competition has run since 2006. The competition takes place each summer at the AAAI or IJCAI Conferences. The event attracts competitors, both academics and hobbyists alike, from countries all over the world. The main focus of the competition is on developing a further understanding of how poker research can benefit artificial intelligence. The competition has four tracks (not all tracks run every year):
  - **Limit Texas Hold'em Poker (2 Players)** – the goal of this competition is to assess agents in game theory applied to large and sequential games.

[18] Official website: http://www.computerpokercompetition.org/

- **Limit Texas Hold'em Poker (3 Players)** – same goal as the 2 player competition but with and even larger game and the multi-player facet, which makes it hard to use Nash-Equilibrium based strategies. Moreover, the existence of a third player greatly reduces the strength of the playing hands.
- **No-limit Texas Hold'em Poker (2 Players)** – the main challenge of this competition is to assess abstraction techniques. Given the fact that No-Limit variants of Poker are much larger, abstraction techniques are essential to make a good game playing agent.
- **Kuhn Poker (3 Players)** – the main challenge of this competition is to assess opponent modelling capabilities in the agents. Given that this variant of Poker is much smaller than the others (as explained in Section 2.5.3), agents must model their opponents to maximize their utility.

— **Bot VS Human competition** – three competitions between agents and very good human players were held. The first competition that opposed poker agents and professional players was in 2007[19]. At this competition, an agent defeated for the first time a professional Poker player in a group of matches in Limit Poker, but lost the series. On the same competition in the following year[20], the competing agent was able to win by a low margin. Finally and more recently in 2015, the competition **Brain Vs AI**[21] was held. It was the first competition in the No-Limit variant between agents and humans. In this competition, humans managed to get more cash but it is stated that scientifically it was a tied competition.

[19] http://webdocs.cs.ualberta.ca/~games/poker/man-machine/2007/
[20] http://webdocs.cs.ualberta.ca/~games/poker/man-machine/
[21] https://www.cs.cmu.edu/brains-vs-ai

3.2 Current Approaches

There are several different approaches for building artificial poker players such as Heuristic-based, Simulation-based, Pattern Matching-based and Monte Carlo Search Tree-based approaches.

In the following sub-sections, the main approaches to create Poker agents will be presented and discussed.

3.2.1 Rule Based

A rule based approach is the most direct and simple approach to start developing Poker agents. It consists in defining the agent's behaviour with a set of conditional **if-then-else** rules, which means selecting an action for a given information set.

A hypothetical rule-based approach is shown on Figure 4. In this particular example the player will fold with a probability of 20%, call with a probability of 30% and raise with a probability of 50%, if the pot value and the hand score is high.

```
Action flopAction(Hand hand, GameState state){
  if(state.pot > 100 && hand.score > 3000 &&
state.numActivePlayers <= 2){
    return new Action(0.20, 0.30, 0.50);
  } else if...
}
```

Figure 4 – A hypothetical rule within a rule-based system for Texas Hold'em Poker (based on Figure 1 in [29]**)**

This approach is very intuitive but has several limitations. First, it requires expert knowledge. Moreover, even with expert knowledge, manually abstracting very large variants of Poker such as Texas Hold'em cannot capture several strategy nuances that expert players use with their intuition, which makes it very hard to formalize.

3.2.2 Simulation Based

Simulation based approaches for Poker game playing consist in generating many random possible match outcomes in order to obtain empirically the statistical average best response for a given game state – Monte Carlo Simulation. Monte Carlo

Simulations rely on simulating the opponents' behaviour when their playing nodes are reached; using random action sampling for every node not only uses a large number of iterations but also could produce not very accurate predictions in games of incomplete information. In order to speed-up the sampling process, the sampling is biased taking into account the opponent profile (pre-established profiles learned from, for instance, game logs [30]) . One of such techniques is called **selective sampling** [31], which combines this with reinforcement learning, which goal is to maximize the information gain – in this work selective bias was introduced in private cards i.e. instead of just considering the opponent's possible private cards, a weighted table with the frequency of those cards is maintained throughout the game and it is filled by all observable actions. A similar technique was also applied to the game of Scrabble [32].

This approach has good practical results in small games, but for very large games such as Poker, the lack of information and observations (only about 10% of the hands provide information about the cards of the opponents) makes the agent play just based on card probabilities, which makes it very predictable.

One particular case of a simulation based approach is the Monte-Carlo Tree Search algorithm (MCTS). The MCTS is a simulation based algorithm that is adapted for sequential problems – it estimates the values of moves by sampling them in a game tree [33]. It consists of applying Monte Carlo Methods on game trees, by selecting random branches of possible outcomes and then selecting the branch that will likely produce the best results. The accuracy of this algorithm greatly depends on the number of simulations: the more simulations, the better the estimate.

The algorithm starts by initializing the game tree, creating a single root node with the current game state. After that, the following steps (see Figure 5 and Figure 6) are repeated a fixed number of times:

- **Selection:** consists on selecting the leaf node of the tree to be expanded;
- **Expansion:** add child nodes to the selected node;
- **Simulation:** game simulation until the leaf node of the game tree is reached;

— **Backpropagation:** the final value of the simulation is stored on the nodes that define the followed path.

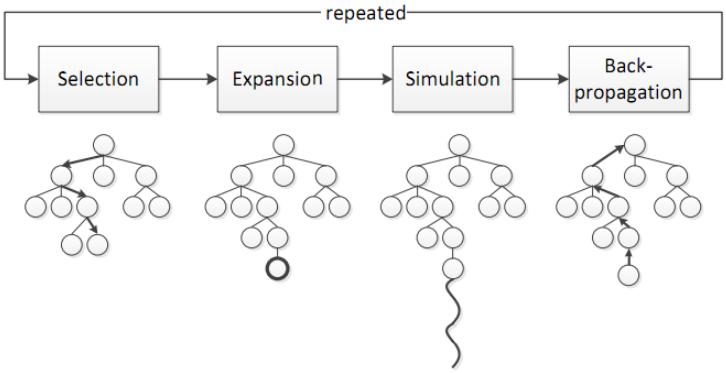


Figure 5 - The Monte Carlo Tree Search algorithm (from [46]).

In this work [33] the author created a complete Poker agent that combines typical player clusters (previously extracted) to predict card holdings and the MCTS algorithm, as explained in Figure 5. The results produced by this algorithm worked well against agents with static strategies but event against those, sometimes problems related with local maxima occurred which prevented the algorithm to properly model the opponents, thus the average winnings were not optimized. Other relevant works about the MCTS algorithm and its applications in Poker are [34], [35].

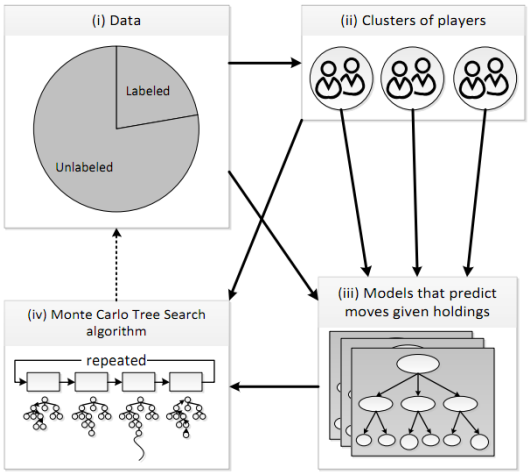


Figure 6 – Poker Agent Architecture that combines MCTS and Clustering (from [46]).

3.2.3 Nash-Equilibrium

A Nash Equilibrium, as defined in Section 2.2.4, is an equilibrium point between a set of mixed strategies, for the game participants, where each one cannot perform better

by changing its strategy, assuming that the opponents maintain their strategies – changing the strategy always results in a worse performance [36]. However, this set of strategies assumes that the players always make the best possible move, which is not the case in Poker, where players are highly fallible. Moreover, a Nash-equilibrium strategy cannot be used to maximize the profit against a particular opponent – it may guarantee a certain utility but it is not able to exploit the opponent. This is especially the case for very weak opponents, where their strategies are very easily exploitable, and using a Nash-Equilibrium against them may produce little profit. Nevertheless, the use of Nash equilibrium strategies in Poker represents a great achievement, especially in heads-up limit poker [24], where the Cepheus agent is taken to be unbeatable. This type of agents are immune to one of the biggest problems of building a Poker agent – they do not require opponent modelling to work. A summary of the application of Nash-Equilibrium profiles to Poker can be found here [37].

Nash equilibrium strategy profiles for two-player Poker or other zero-sum sequential games can be generated using linear programming techniques (such as Simplex [38]) when the game search space is small. However, due to the large number of information sets of common Poker variants, this is currently unfeasible. Instead, what is usually computed is a Nash-equilibrium over an abstracted version of the game (joining information sets into the same bucket) producing therefore a ε-Nash Equilibrium, resulting in an optimal set of strategies within the abstraction – see the summarized steps in Figure 7. The Nash-equilibrium strategy profile on the smaller abstract game has to be translated to the real game. This usually happens when bet amounts are abstracted in No-Limit Poker – the translation step will usually return a range of possible bet values and a random bet value will be generated. One of the problems of this approach is that we are not solving the exact same game, which can add noise to the strategies. Moreover these strategies are still exploitable, and they could even become predictable, because they highly depend on the used abstraction. If the opponent discovers the abstraction that is being used, he or she can easily exploit the agent.

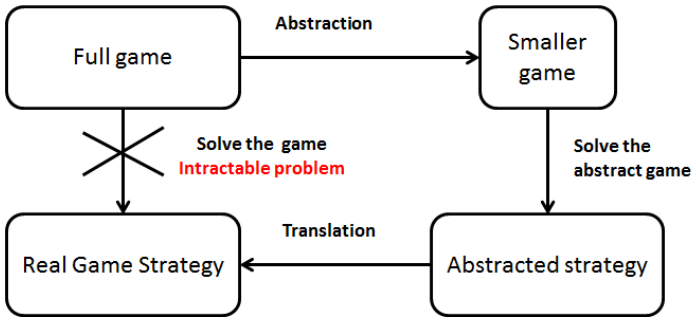


Figure 7 – Reducing the size of a large game (adapted from [36])

There are some ways to improve a strategy based on a set of Nash-equilibrium strategy profiles:

- **Best Response** – the best response is the definition of a counter-strategy for a given strategy *s* – it always makes the statistically best decision against *s*. Computing a best response is computationally expensive, so usually only an approximation is calculated (could be done with Monte-Carlo methodologies) – this assumes that the strategy *s* is completely open (we can see the information sets and the decision on strategy *s*) because it is not possible to compute a best response online. Computing the best response to a set of Nash-equilibrium strategies leads to the **exploitability**. The reason behind this is that a true Nash-Equilibrium strategy has 0 exploitability: this way, we can compute how far we are from a perfect Nash-Equilibrium. A formal description of this algorithm is presented in [36]. One possible way of using the Best Response during online gaming is to use a technique called **Frequentist Best Response** – creation of an opponent model for the abstract game by observing the logs on unabstracted data, thus creating the action selection tuples in form of frequencies (e.g. Fold 20%, Call 40% and Raise 40%). This assumes having the logs with lots of data of the opponent before having matches with him / her, which is not feasible. However, experiences with simple agents proved that this works, at least in theory [39].

- **Restricted Nash Response** – this method [36] combines the Nash Equilibrium and the Frequentist Best Response approaches. It is composed by several strategies that are frequentist responses to several types of opponents {so_{t1}, so_{t2}, so_{t3}… so_{tn}} and an additional Nash-Equilibrium strategy *SE*. The reason

behind this is that it usually takes a lot of iterations (games) until we find out in which profile t_{player} we can assume that our opponent is playing with. So, while we do not know this, we can use a Nash-Equilibrium strategy (which is more conservative) to keep in the game without losing too much cash / profiting less cash. As it was written before, besides being a very robust strategy against a random opponent, a Nash-Equilibrium strategy does not maximize profit against particular sets of opponents. The combination of these two methods solves one of the Nash-Equilibrium strategy issues. However there are some limitations with this approach. Making models is a very hard task, as it requires lots of domain knowledge. In case our models are inaccurate or incomplete due to a limited number of observations, this kind of approach could perform poorly.

- **Data Biased Response** – this approach [40] tries to address some of the limitations found in the restricted Nash response approach. The improvement was made by including not only the frequencies of actions but also the probability of reaching a given information set – if we are playing against the opponent with an information set which is not likely to be reached, we can switch to another strategy (even to pure abstracted Nash-Equilibrium strategy), solving partially the problem of having limited observations. Another advantage of this methodology is that it actually does not need full opponent profiles but instead opponent profiles per information set.

The current state of the art algorithm for computing a Nash-Equilibrium (see Section 3.2.4) for a large sequential game is Counterfactual Regret Minimization (CFR).

3.2.4 Counterfactual Regret Minimization

Counterfactual Regret Minimization (CFR) is an algorithm that is used to find approximate Nash-Equilibrium solutions for very large sequential games. This algorithm is based on the concept of **counterfactual regret** first defined by Zinkevich et al. in [41]. **Regret** is a measure for decisions – it is the difference between the utility of any action and the utility of the action that was actually chosen. To better illustrate this definition, consider the Rock-Paper-Scissors game (see extensive-form and normal-form in Figure 1 and Table 1, respectively). Consider the follow events:

- Player A chooses Rock
- Player B chooses Paper
- Player B wins, thus the utility for player A is -1 and for player B it is +1.

The regrets for player *A* are:

- Rock: 0, because it was the action actually selected by player *A*.
- Paper: 1, because he lost 1 point, but if he had chosen Paper he would tie and therefore not lose any point.
- Scissors: 2, because he lost 1 point, but if he had chosen Scissors he would win the game and would have 2 more points of utility.

The counterfactual regret is obtained by using the **regret matching technique**, i.e. by normalizing the accumulated positive regret of the simulated games and weight it with the probability of the opponent reaching that information set (in this case the probability is equal for every information set: 1/3). Therefore, after this first match the counterfactual regret would be:

- Rock: 0
- Paper: 1/3
- Scissors: 2/3

The regret matching technique leads to a best response. CFR is recursive algorithm that consists of having two or more agents using the regret matching technique against themselves in several consecutive iterations. Since each agent is adapting to its opponent they will both converge to an equilibrium point where both of their strategies are in a Nash-Equilibrium. The CFR algorithm is only proved to mathematically converge for two player games [42]. The steps for performing the CFR algorithm are (see Figure 8 for an example information set):

- Compute the expected utility of each action
- Calculate the counterfactual regret for each action
- Update the accumulated counterfactual regret
- Compute the new strategy probabilities proportionally to all positive counterfactual regret values.

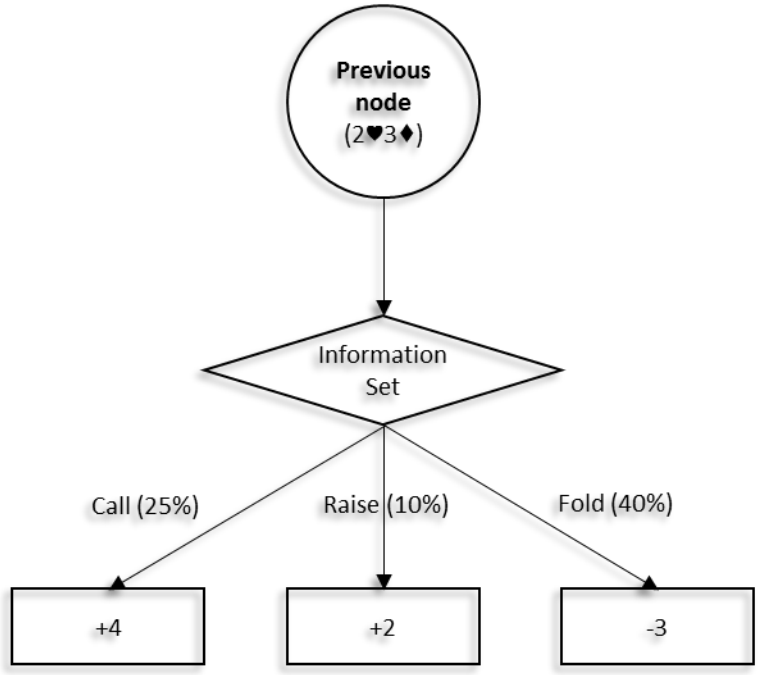


Figure 8 – Information set example (adapted from [36]**)**

One important thing is that the probabilities of all actions should be initialized with an uniform probability distribution (in case of Poker (1/3, 1/3, 1/3)). Using a random distribution may cause that some important information sets will never be reached.

There are several variants of CFR built over the years:

- **Monte Carlo CFR** – CFR is a recursive algorithm that visits all game nodes. The number of game nodes in Poker is huge, therefore the game is abstracted to reduce the amount of game nodes. In order to reduce the amount of abstraction needed, this technique [43] combines the CFR algorithm with MCTS. Instead of expanding all actions nodes, some of the nodes are randomly ignored when calculating the counterfactual-regret. Results showed that it usually performed better than the regular CFR. There are three heuristics to ignore action nodes: *opponent-public chance sampling, self-public chance sampling* and *public chance sampling*. In [44] it is demonstrated that the **Public Chance Sampling** version performs better.
- **CFR-BR** – in this version of the algorithm, instead of being executed with two agents that know the abstracted game, CFR leaves one of the agents unabstracted. For this to work, it is assumed that the unabstracted agent uses a best-frequentist response on each iteration and we try to find the optimal strategy for the abstracted agent. Like in Monte-Carlo CFR, abstraction techniques are also used, leading to a not exact best response. With these improvements, the exploitability of the strategies produced by

CFR-BR are much lower than the ones produced by regular CFR or Monte-Carlo CFR [21].

- **CFR+** – this version of the algorithm [24] uses a technique called *regret matching plus* which constrains the counterfactual regret computations to be non-negative. This version of CFR does not require abstraction, thus leading to the very first solution of Limit Texas Hold'em Poker – with parallelisation, 4800 CPUs and 68 days of intensive computation.

3.2.5 PokerLang

Due to its stochastic nature, Poker players use specific strategies for similar game conditions. A strategy is used under certain information sets that are described by specific visible game conditions such as the card probabilities (hand strength), player's cash, number of opponents, playing order, among others. These are known as the game features – characteristics of the information set that influence player decisions.

A strategy *S* can be conceptualized as a set of tactics. A tactic $t \in T$ is a mapping between a set of information sets and a set of actions:

$$t: I' \rightarrow A' | I' \subset I \wedge A' \subset A$$

EQ8

I' and A' represent two types of game abstraction: information set abstraction and action abstraction (respectively). This is done by transforming F into F', where the features of F' are simplified so that $|I'| < |I|$. The information set abstraction is particularly essential because Poker has so many information sets that it would not be possible, with current hardware, to store the corresponding action for each one. For a similar reason, action abstraction is also handy; in No-Limit Poker there is a continuous interval of possible decisions. Usually this interval is discretized into a fixed number of possible decisions: fold, call, intervals of raise values and all-in (betting the remaining cash). Using a fixed number of decisions simplifies search-tree strategy based algorithms, because it greatly reduces the horizontal and vertical expansion of the decision tree by reducing its branching factor.

In order to specify these concepts, high-level language was created – *PokerLang* – whose syntax and grammar was based on Coach Unilang. Its specification is

described in [45]. The generic approach of this language allows for its easy adaptation to other domains.

The language root starts by defining the concept of strategy: a strategy is a set of tactics each of which is a tuple composed by an activation condition and a behavior for that tactic. The activation condition consists of abstracting decision points or information to define I'. They correspond to a set of verifications of the visible game features (through evaluators) or predictions about uncertain events (through predictors). A tactic's behaviour is the procedure followed by the player when the activation condition is met (the behavior itself has a second layer of verifications that can abstract the information set even further). The tactic's behaviour could be either user-defined or language predefined (based on common expert tactics). In the next sub-sections we describe PokerLang's main language concepts. Below the main elements of this language are presented, in the BNF notation.

```
<STRATEGY>::= {<TACTIC>}

<TACTIC>::= <ACTIVATION_CONDITION> <TACTIC_BEHAVIOUR>

<ACTIVATION_CONDITION>::= {<EVALUATOR>}

<TACTIC_BEHAVIOUR>::= <PREDEFINED_BEHAVIOUR>|<BEHAVIOUR>

<PREDEFINED_BEHAVIOUR>::= loose_agressive|loose_passive|

tight_agressive|tight_passive

<BEHAVIOUR>::= {<RULE>}

<RULE>::= {<EVALUATOR> | <PREDICTOR>} <ACTION> <VALUE>

<ACTION>::={<PREDEFINED_ACTION><PERC>|
      <DEFINED_ACTION><PERC>}
```

Language: BNF

To allow the easy creation of Poker-Lang document, an interface was also created. This interface is called *PokerBuilder* and its appearance can be seen on Figure 9. With a smooth interface and simple features, *PokerBuilder* is accessible to any user that understands the main concepts of poker.

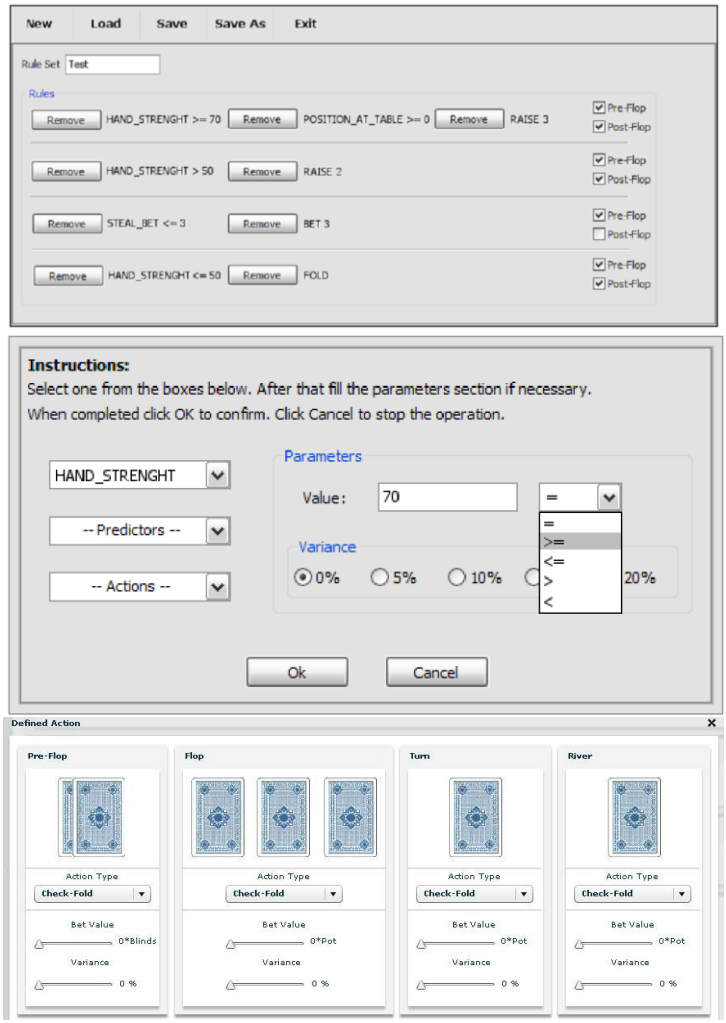


Figure 9 – Poker Builder (from [45]**)**

3.2.6 Other approaches

Several other approaches were used for building poker playing software agents but with less scientific significance. However, some of the agents developed with such approaches actually got some very good results in practice, often surpassing more theoretical approaches (with emphasis for a Case Based Reasoning Bot that got 2nd and 3rd places at the Computer Poker Competition even against Nash-Equilibrium agents). These approaches are:

— **Miximax** and **miximix**– these algorithms were presented on [46], [47] and they can be considered to be versions of the minimax algorithm adapted to incomplete information games. These algorithms are adaptive: they assume

the existence of opponent profiles with online information from the games assigning one of the profiles to the current opponents. This will enable to better estimate the probabilities of all actions on a given information set thus permitting the agent to traverse the game tree with much more accuracy. This approach is closer to what real players do, thus having the disadvantage of needing a strong starting database in order to properly estimate how the opponents will proceed in order to select the best possible action. Another problem is the predictability of this kind of approach, after the opponent learns which models we know.

- **Teams of Computer Programs** – This approach is in reality an aggregation of several methodologies – it consists of using several agents at the same time. Each agent has strengths and weaknesses. When facing an unknown opponent, there is no information regarding which strategy should be used. An approximation to a Nash-Equilibrium is not likely to lose, but it is always possible and it was already explained that the goal is to earn as much as possible. Therefore using heuristics to switch strategy during the game (if one is not performing well) is a good option, with very good results demonstrated empirically [30], [48]. The agent that is responsible for changing the strategy is called *coach agent*. It observes the conditions of the game and decides which of the game playing agents is going to play.
- **Pattern Matching Methods** – pattern matching methods consist of creating agents that adapt their strategies based on past experience. The idea of these methods is to model the game, by defining which game variables represent a strategy and then build statistical models of past game data based on those variables. For instance, the work described in [18] showed that it is possible to build a Poker agent that behaves like a human player, using supervised learning methods. The author defined game variables and then used game logs of past games between human players to copy their strategies. The agent is also able to combine different tactics from different human players. This way, the agent does not have a static strategy; therefore it can confuse the opponent modelling mechanics of its opponents. Another similar approach [49] used the expectation-maximization (EM) algorithm to build cluster models, where a

cluster model is a mixture model of players. This approach focused on learning quickly instead of learning accurately because if a player wants to win against an opponent that it has never seen, it has to learn fast before losing its chips. This approach performed well in short term games, outperforming agents based on Bayesian methodologies.

- **Case-Based Reasoning (CBR)** – is a group of learning algorithms that consist on having a set of previously observed cases as a knowledge base to aid in the decisions during the game. In the case of Poker it uses classified game logs as a knowledge base, with each play classified as being good or bad. When the agent has to decide which action it is going to take it searches on the knowledge base for the case that more reassembles the current game state. The decided action is similar to the one that was taken in the past (if the action was good in the past). The most successful applications of CBR to Poker were Casey[50], Casper[51] and Sartre[52]. They got very good standings in the ACPC competition (including a 3rd place in No-Limit).

3.3 Poker Agents

The number of poker software agents has been increasing in the last years. Many have been created during, and as a part of, academic research but as this research matures and becomes widespread, so does the number of individuals tackling and researching on this subject.

This section provides a brief description and resources for the most popular poker agents at this time, with emphasis on the ones that participated in the Annual Computer Poker Competition or that produced some academic results. There is not much info about other potential agents that are being used in Online Poker. The limitations imposed by casino clients in their use makes them being hidden to the general public. However, there is strong belief among Poker professionals that bots are playing everywhere online.

Table 5 – Summarized description of some notable Poker Agents

Year	Name	Type
1997	**Loki**	**Rule Based**
Loki [53] was the first agent made by the CPRG. Loki uses a rule based approach which was made by game experts. Its decisions were mainly based on profile opponent models based on expert knowledge and Effective Hand Strength computation. This approach was not very successful in online matches (on IRC) against low/mid-level opponents. Also it was not very accurate with opponent modelling and it had a huge exploitability. When CFR agents emerged, Loki became quickly deprecated.		
1999	**Poki**	**Rule Based**
Poki [46] is the new version of Loki, completely revised. It was the first agent to feature the miximax and miximix techniques. This version had a revised opponent modelling system which made it rather successful in IRC matches. It also won the 2008 ACPC 6 players limit competition – even against CFR agents. This happened because currently CFR agents are not yet proficient in multiplayer. In games with lower number of players, this agent can be easily beaten by a CFR. Even so, it is still an important agent, with potential applicability for online game playing against humans.		
2002	**PsOpti/Sparbot**	**Nash-Equilibrium / Linear Prog.**
PsOpti [54] was the first one to use a Nash-Equilibrium based approach with abstraction. This agent only played Texas Hold'em Pre-Flop. Even so, tests indicated that it outperformed some human players (on IRC) and all agents developed until then. However, the winning rates of this agent were low, because it did not possess any opponent modelling capabilities. It participated in the ACPC competition in 2006 and won the Head's on Limit competition under the name of Hyperborean 06. Its code was also included in the Poker Academy software under the name of Sparbot.		
2003	**Vexbot / BRPlayer**	**Adaptive game tree search**
Vexbot [55] uses context tree data structures to store the opponent models. These models disregard chance nodes and only store betting sequences. It was the bot that firstly stored decisions with abstracted game betting sequences as the key for retrieving the probabilities to play on given information sets. This agent was also the first to be able to detect weaknesses in a Nash-Equilibrium strategy by exploiting the older PsOpti versions.		
2006	**Hyperborean**	**CFR / Nash-Equilibrium**
Hyperborean [41] first emerged as a team of poker programs composed by all PsOpti agents. It participated and won the first ACPC competition on Limit Poker variant. Furthermore, this agent marked the introduction of the CFR algorithm as the state of the art algorithm to create Near-Equilibrium agents.		
2007	**Polaris**	**CFR / Restricted Nash Response**
Polaris [36] innovated by using the restricted Nash response technique described in Section 3.2.3. It was the first game playing agent to have reported wins against some of the best human Poker players in the first Human VS Machine competition.		
2007	**Hyperborean (No-Limit)**	**CFR / Restricted Nash Response**
Hyperborean (no-limit) [56] was the first bot to apply the CFR and Nash-Equilibrium approach to a No-Limit Poker version. This agent was the first that needed to explore an extra step required for game abstraction, called translation. The translation was		

required to correct the betting amounts between the abstracted and the real game because in No-Limit Poker the betting amounts are continuous.		
2009	**Hyperborean (Ring)**	**Data biased response**
Hyperborean (ring) [42] was the first one based on the CFR algorithm that performed well in multiplayer games. Despite the lack of theoretical guarantees that CFR produces Nash-Equilibrium strategies for multiplayer games, this agent is the proof that the generated strategies are still very robust for game play.		
2015	**Cepheus**	**CFR+**
Cepheus was the first agent [24] to solve the Limit Texas Hold'em variant of Texas Hold'em without abstraction. Despite having the problems of still not being able to optimize winnings against specific players, it is unbeatable in the long run, because its exploitability is almost zero. The main issue is that the used algorithm (CFR+) requires a lot of computational power and memory to deal with such large variants of Poker.		
2006	**Casey**	**Case Based Reasoning**
Casey [50] is a case based reasoning bot that starts off with an empty knowledge base. It starts playing with random decisions and recording all of them. The more it plays the more it learns. This version made some erratic assumptions of the game without an initial training period.		
2007	**Casper**	**Case Based Reasoning**
Casper [51] is another Poker agent that can play in a Full Texas Hold'em Poker table which demonstrates the usefulness of these approaches over more theoretical approaches when it comes to multiplayer Poker. This agent does not learn from scratch like Casey, it uses game logs from playing against Poker Academy to learn new cases to make its decisions. This agent did well against the agents at Poker Academy and also against humans with fake money, but it did not do as well against humans in real money tables with very small stakes.		
2009 / 2010	**Sartre**	**Case Based Reasoning**
Sartre [52] is the updated version of Casper but Nash-Equilibrium based agents. In 2010 it got the 3rd place in the Limit competition of the ACPC and the 2nd place in the No-Limit variant.		
2006 / 2007	**GS Family**	**Nash-Equilibrium**
GS is bot similar to PsOpti that uses the Game Shrink system. That algorithm, given a description of the game tree is capable of generating a Near-Nash Equilibrium solution using lossless or lossy abstractions. This agent also combines offline and real-time game solving, using offline learning for the Pre-Flop and Flop rounds of the game and for the other the solution is computed online.		
2007 / 2015	**Tartanian**	**Evolutionary game theory**
Tartanian is a group of game playing agents that innovated by their abstraction techniques on No-Limit versions of Poker. It is also the first reported agent that applies EGT to Equilibrium solution learning, by breeding and merging different equilibrium profiles to speed-up reaching a more stable solution. It placed 2nd at the 2007 ACPC competition.		
2010	**HoldemML**	**Pattern Matching**
HoldemML is a group of No-Limit Texas Hold'em game playing agents that used pattern matching to build their behaviour from game logs [48]. The results demonstrated that these agents imitated well their human counterparts but the lack of		

available data makes this approach very hard to be feasible for online play.		
2015	**Lucifer Hold'em**	**Iterative CFR / Teams of CFR**
This is an agent that used, for the first reported time, a non-recursive version of the CFR algorithm. Despite the lack of training time (1 hour against the average of 2 weeks) and computational resources it still could get the 9th place out of 16 at the Head's on Texas Hold'em Limit Competition. Some of the used techniques are described in Chapter 6.		

3.4 Hand Rank Computation

A Poker hand is a set of five cards that expresses the player's score. Let's consider the same notation described on Section 2.5.4. Being D the set of all cards in the deck, P_i the set of pocket cards of a particular player *i* and *S* the set of community cards so that $\mathrm{P}_1 \cup \mathrm{P}_2 \cup \ldots \mathrm{P}_N \cup \mathrm{S} \subseteq \mathrm{D}$, and $\mathrm{P}_i \cap \mathrm{S}$ for any *i* is equal to $\varnothing$. Thus, the score function is defined as $sc\colon [\mathrm{D}]^5 \to \mathbb{N}$. For a particular player, the hand h_i is the union of the pocket cards and the community cards ($\mathrm{P}_i \cup \mathrm{S}$). Thus, the player's score is given by the rank function, as follows (EQ9):

$$Rank(h_i) = \max(\{sc(x)\colon x \in [\mathrm{P}_i \cup \mathrm{S}]^5\})$$

EQ9

The possible hand ranks are from stronger to weaker (see with more detail in Section 2.5.4.1): Straight Flush (sequence of same suit), Four of a Kind (4 cards with same rank), Full House (Three of a Kind + Pair), Flush (5 cards with same suit), Straight (sequence), Three of a Kind (3 cards with same rank), Two Pair, One Pair (2 cards with same rank) and Highest Card (not qualifying to other ranks). Examples of each rank are demonstrated in Table 3. These ranks are not equally valued. Each rank has sub-ranks essentially based on the score of the top cards (e.g.: a pair of aces scores higher than a pair of queens). In total, there are 7,462 possible sub-ranks in Texas Hold'em Poker.

A poker hand rank evaluator is a software program that computes the value of the rank function, partially computed by the score function $s\colon [\mathrm{D}]^5 \to \mathbb{N}$. In Texas Hold'em Poker this evaluator receives as parameter the set of cards P_i + S, where $|\mathrm{P}_i| = 2 \wedge |\mathrm{S}| \in \{5,6,7\}$. The evaluator returns a natural number representing the relative value of that hand (typically from 0 to 7,461, where 7,461 corresponds to one of the top scored Straight Flushes). The charts in Figure 10 show the relative

frequencies of each score for Flop, Turn and River, respectively $|S| = 5$, $|S| = 6$ and $|S| = 7$. The horizontal axis represents the hand ranks (ordered) and the vertical axis is the relative frequency of that hand.

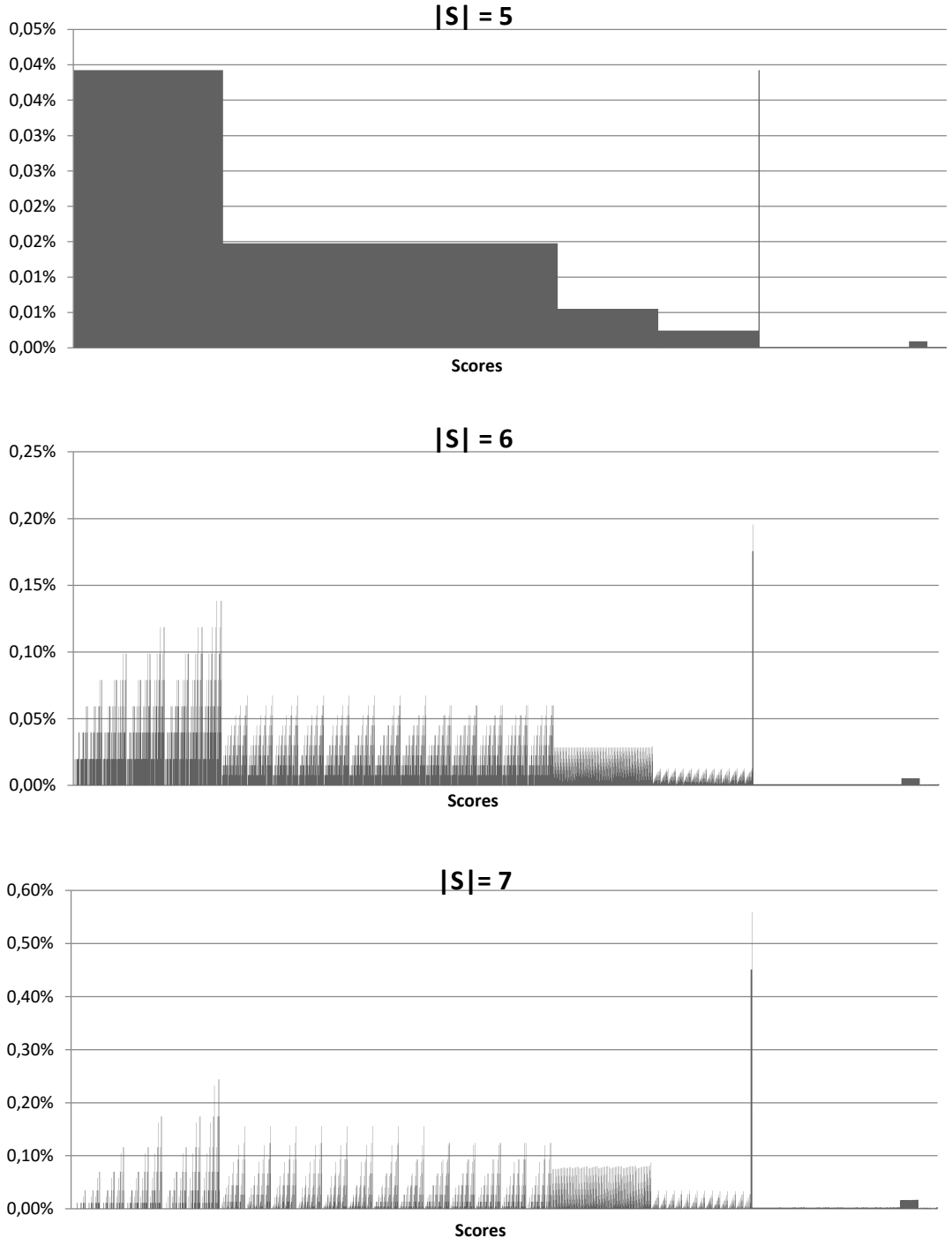


Figure 10 – Hand rank distributions in Flop (top), Turn (middle) and River (bottom)

It is possible to observe a stair step layout in the first chart ($|S| = 5$). Each stair represents a hand name in Table 3. It is also possible to observe large peeks near the

end of each chart. They represent the straight hands, because there are plenty of ways of combining 5 cards to score a straight, but only 10 possible types of straights (Five high, Six High …).

To compute the probability of success of a given hand – odds – it is usually necessary to compute several hand ranks before. For instance, the odds calculation methodologies presented in the next section require the computation of hand ranks.

Programming an algorithm to determine the hand's rank is a relatively trivial task. This can be done using a naïve approach, i.e. using an algorithm that intuitively makes sense and that is humanly readable. Naïve hand rank evaluators usually consist of the following steps:

- Sort the hand by card value (deuce has the lowest value and ace has the highest);
- Iterate through the hand, collecting information about ranks and suits of the cards;
- Make specific tests to check, iteratively, if the hand is of a certain rank, starting at the higher ranks.

One example to illustrate this idea can be found in Figure 11. This example does not consider the whole set of Texas Hold'em rules.

```
Function HandRank(Hand) {
    Sort(Hand);
    If IsStraightFlush(Hand) Return 9;
    If IsIsFourOfAKind(Hand) Return 8;
    If IsFullHouse(Hand) Return 7;
    If IsFlush(Hand) Return 6;
    If IsStraight(Hand) Return 5;
    If IsThreeOfAKind(Hand) Return 4;
    If IsTwoPairs(Hand) Return 3;
    If IsOnePair(Hand) Return 2;
    Return 1;
}
```

Figure 11 – Hypothetical Naïve Hand Rank Evaluator

The problem with naive evaluators resides in their efficiency, which is important because the rank evaluator is used by a hand odds evaluator several times per

computation. The solution to this problem resides in top-down dynamic programming algorithms in order to speed up the rank function. The next subsections will present some developed approaches to solve this issue.

3.4.1 Pokersource Poker-Eval

Poker-Eval is a C implementation of a Poker Hand rank evaluator [57]. As described at the beginning of this section, given a hand, this evaluator returns a natural number that represents the hand score. This evaluator uses a naïve approach and, to the best of our knowledge, the fastest one.

The main advantages of this evaluator are its architecture which supports multi Poker variants, multi-platform usage, since there are wrappers for other programming languages and its low memory usage when compared to look-up table based approaches. The main issue of this evaluator is its low level API which makes it hard to use by programmers.

3.4.2 Cactus Kev

The Cactus Kev's 5-Card Evaluator [58] is a system to compute 5 card hand rank. The idea behind its algorithm is the construction of a pre-computed look-up table with every possible rank. However, since the number of possible 5 card sequences is ${}_{52}P_5$, the size of the table would be about 2.5 GB of memory (considering 8 bytes to store the hand and its rank).

To solve this problem one can group similar hands (same cards, different order), resulting in $\binom{52}{5}$ hands, making this approach feasible (the size of the new look-up table would be about 20 MB). However, this solution requires sorting the hand cards before accessing the look-up table, wasting additional CPU time. To solve this, Cactus uses a 32 bit integer representation of the cards (Figure 12).

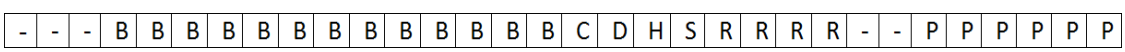


Figure 12 – Cactus Kev's card representation

P (6 bits) represents the value of a card in a form of a prime number, with the following values Two – 2; Three – 3; Four – 5; Five – 7; Six – 11; Seven – 13; Eight – 17; Nine – 19; Ten – 23; Jack – 29; Queen – 31; King – 37; Ace – 41. The reason behind this

decision resides the fact that the multiplication of two prime numbers always generates a unique value. This allows for avoiding the step of sorting the hand cards, saving CPU time. Therefore, the product of these values can be used to index the hands.

R (4 bits) represents the rank of the card (Two – 0; Three – 1; Four – 2; …). **CDHS** represents the card's suit mask, where one of the bits is activated (C if the card is Clubs, D if the card is diamonds …). The **B** (13 bits) represents the card's rank mask, where the first bit is activated when the card is a Two, the 2nd bit is activated when the card is a Three, and so on.

Three look up tables are used in this evaluator: **flushes** (the ranks of all flushes and straight flushes hands), **unique5** (the ranks of all hands with cards with different ranks) and **values** (the remaining cards). To build the look-up tables, a naïve evaluator is required.

To find the value of a certain hand, the three tables are consulted. Assuming the cards of the hand are labelled as C_1, C_2, C_3, C_4 and C_5, Cactus first verifies if the hand is a flush: Index = C_1 AND C_2 AND C_3 AND C_4 AND C_5 AND 0x0F00

For the calculated index, the table can either return the value of the hand or 0, if the hand is not a flush or a straight flush. The next step is to verify if the hand belongs to **unique5** by calculating the following way: Index = (C_1 OR C_2 OR C_3 OR C_4 OR C_5) >> 16

Once again, if the value of the table at the calculated index is 0, we have to look for the result in another table. The final index uses the described prime number strategy.

$$Index = \prod_{i=1}^{5} (C_i \; AND \; 0xFF)$$

EQ10

The problem of using this index system is that it would result in a very large look-up table of size $41 \times 41 \times 41 \times 41 \times 37 = 104{,}553{,}157$. The author of this technique

solves this problem by storing the indexes in a binary search tree for fast hand value retrieval.

The main limitation of this hand evaluator is that it can only be used to evaluate 5-card hands. This means that to use it in Texas Hold'em (which needs to evaluate 7-card hands in the River round), the function has to evaluate all possible 21 combinations of 5 cards to determine the hand value.

3.4.3 Paul Senzee

Paul Senzee's hand evaluator is an improved version of Cactus Kev. However, instead of using a binary search, Senzee uses a pre-computed perfect hash table.

A perfect hash table guarantees no collisions in the storage of the hands' values. Also it allows for acquiring the values in constant time instead of the $O(\log n)$ complexity of the binary search. The used hash function was based on [59]. This approach produced a time improvement factor of about 2.7 times [60].

Another advantage of Paul Senzee's evaluator is that it provides 7 card hand evaluation (River round), without having to compute all possible ranks (21) to pick the best one.

Paul Senzee's 7 Card Evaluator also uses a pre computed hand table to quickly determine the integer value of a given 7 card hand. For 7 hand cards lookup, Paul represents each hand with a 52 bit string, where each bit represents an activated card. The total number of activated bits is 7, representing a 7 card hand.

If unlimited memory was available, it would be possible to index the resulting rank value into an enormous and very sparse array with 2^{52} entries of about 9 petabytes of memory (9 million gigabytes). To solve this problem, Paul Senzee's developed a hash function that turns the hand value into an index between 0 and roughly 133 million and, by using the Cactus Kev's evaluator, it is possible to produce a 266MB lookup table. The author of this approach does not provide information about the hash generation code. The main limitation of the 7 card version of Paul's evaluator is that only supports 7 cards (it does not support Flop and Turn rounds).

3.4.4 TwoPlusTwo Evaluator

TwoPlusTwo evaluator is a lookup table Poker hand evaluator that uses a table of 32,487,834 entries with a total size of ~130 MB [61]. The TwoPlusTwo Evaluator is extremely fast and probably the fastest hand evaluator there is. This is because the ranks of the hands are stored in a non-sparse array with low redundancy.

To store the hands, the implementation of this evaluator is based on a direct acyclic graph of seven layers, with each edge representing a card value. Therefore, each node of this graph represents a card sequence and it links to nodes with the same card sequence but with one extra card. In the final layers (5, 6 or 7), the node contains the hand value. This representation would require $\frac{52!}{(52-n)!}$ positions for layer n. However, the author of this method grouped similar hands, since the order and the suits (except for flushes) of the cards do not matter for the hand score. Using this structure, to get the value of a given hand, only one lookup per card is performed. For instance, the following function will compute a 7 card hand value, being **HR** the lookup table.

```
Function Rank(Hand) {
 Return HR[HR[HR[HR[HR[HR[HR[53 + Hand[0]] +
Hand[1]] + Hand[2]] + Hand[3]] + Hand[4]] +
Hand[5]] + Hand[6]]
}
```

Figure 13 – Using the TwoPlusTwo evaluator

There is also an improved version of this evaluator by Jan Varho. Varho method splits the entire lockup table into 7 distinct tables, each one of them representing a lookup layer. This way, Varho was able to save memory by using short numbers (16 bits) to store the final layer. The total size of all tables is now about 80 Mb [62].

3.5 Hand Odds Computation

Evaluating the odds of a hand consists of measuring its quality at any state of the game. This section describes how to compute the probability of a complete hand (hands with 5 or more cards) being successful at Showdown (last round of Poker where the players show their cards and the winner is decided). By evaluating the hand it is

possible to determine the probability of winning or losing the current game. This knowledge can be used to inform the agent's decision of either fold the hand or play it, as well as to assess the probability of success and the risk that the agent is facing. Computing the hand odds may consider the following variables: Pocket cards; Number of opponents; Community cards; possible community cards to come and possible opponents' cards.

The hand evaluation method typically returns a probability. If it returns the lower limit, this means that the hand will lose regardless of future events in the game, unless the player uses deception to bring opponents to forfeit. Conversely, obtaining the upper limit from the hand evaluation function means that victory (or at least draw) is mathematically assured – the only way of losing is to unwisely fold the hand.

3.5.1 Hand Strength

The hand strength [17] is the probability of the current hand being the best if the game reaches a showdown with all remaining players. It consists of enumerating all possible hands that an opponent can have and checking if the agent's hand is better than the hands in the enumeration. By counting the number of times the player's hand is found to be better, it is possible to measure the quality of the hand. Using Section 2.5.4 terminology, the hand strength (*HS*) for a given number of opponents *n* is given by:

$$Ahead(h_i) = |\{\forall \mathrm{x} \in [\mathrm{D}\backslash \mathrm{P}_i]^5 : \mathrm{sc(x)} < \mathrm{Rank}(h_i) \wedge \mathrm{x} \supseteq \mathrm{S}\}|$$
$$Tied(h_i) = |\{\forall \mathrm{x} \in [\mathrm{D}\backslash \mathrm{P}_i]^5 : \mathrm{sc(x)} = \mathrm{Rank}(h_i) \wedge \mathrm{x} \supseteq \mathrm{S}\}|$$
$$Behind(h_i) = |\{\forall \mathrm{x} \in [\Delta\backslash \mathrm{P}_i]^5 : \mathrm{sc(x)} > \mathrm{Rank}(h_i \mathrm{h}) \wedge \mathrm{x} \supseteq \mathrm{S}\}|$$
$$HS_n(h_i) = \left(\frac{Ahead(h) + \frac{Tied(h)}{2}}{Ahead(h) + Tied(h) + Behind(h)} \right)^n$$

EQ11

The Hand Strength may be used in any round of the game. However hand strength does not address the possibility of the hand improving in subsequent rounds of the game, which is possible because in Texas Hold'em new cards are revealed at the start of every round (community cards). This issue is addressed by the Hand Potential Formula [17] which sums up possible hand strengths in subsequent rounds (described in Section 0).

In [63], the authors suggest it is possible to combine the hand strength algorithm with opponent modelling in order to calculate the hand strength taking into account the opponents. For this purpose, the proposed algorithm would use $Remain' = [\mathrm{D} \backslash \mathrm{P}_i \backslash \partial]^5$ where ∂ is the set of cards that the opponent probably does not have, given that $(\mathrm{P}_i \cup \mathrm{S} \cup \partial) \neq \mathrm{D}$, and $(\mathrm{P}_i \cap \mathrm{S} \cap \partial) = \varnothing$. This approach was successfully tested in Texas Hold'em heads up games.

In Figure 14 it is possible to observe the heat map for the average hand strength against 1, 2, 3 or 4 opponents. For the following heat map, the colours have the following meaning (blue – (probability ≥ 90%); purple (probability ≥ 70% and < 90%); green (probability ≥ 50% and < 70%); red (probability < 50%).

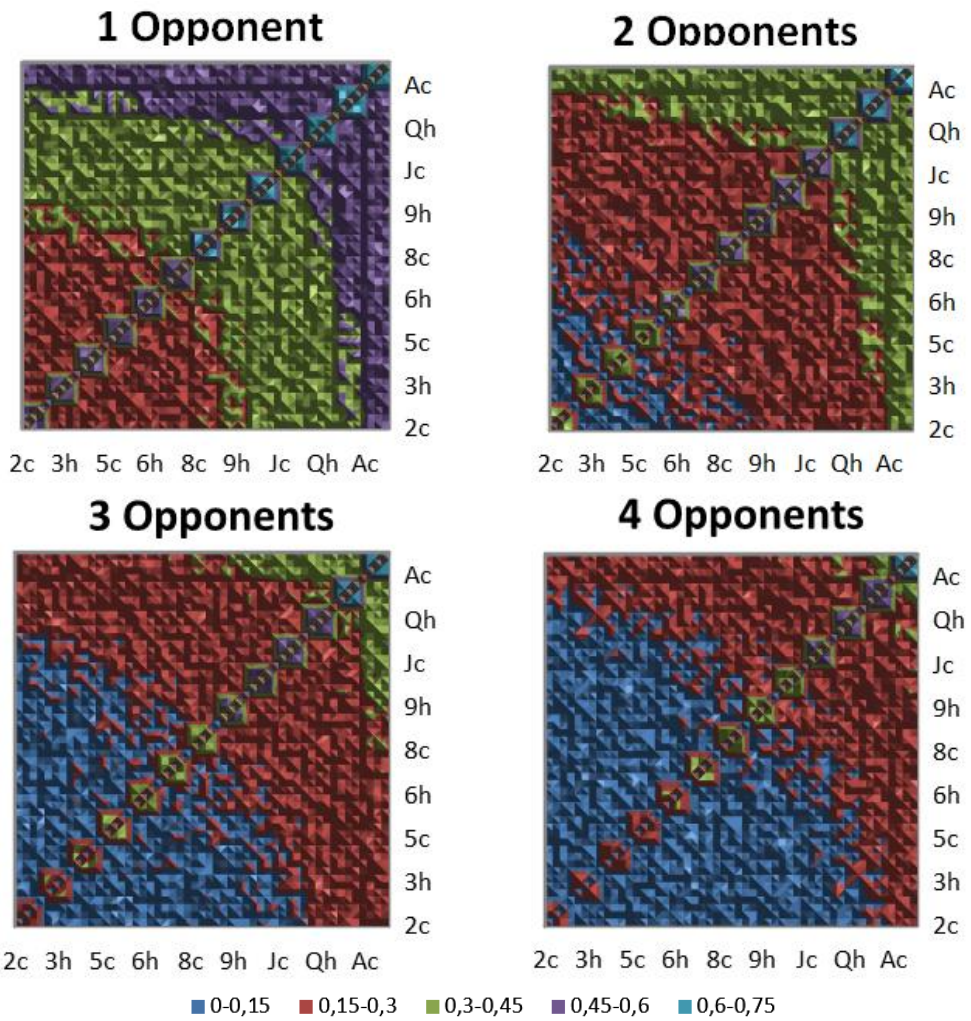


Figure 14 – Heat maps for hand strength against a variable number of opponents. The horizontal and vertical axis represent a card and the 'heat' is the value of the average hand strength for the pair of cards.

As expected, it is possible to notice that there is a larger concentration of high hand strength values next to the higher cards. Moreover, as the number of players increase, the area of high hand strength values decreases. This means that players should be more careful when playing against a higher number of players, since there is a greater probability of one of them having a better scored hand.

3.5.2 Hand Potential

Hand potential [17], [63] is an algorithm that calculates the possible evolution of the hand quality throughout the game. In Texas Hold'em, when the game reaches the Flop round, there are still two more deck cards to be revealed. This means that the current hand rank may improve, since the hand is composed of the set of five available cards (pocket or community cards) that has the highest rank among all available cards. This is an extension of hand strength, but instead of only considering the current available cards, it considers the possible community cards that have not been revealed yet. This also considers that the opponents' hands might improve as well. Hand potential has two components:

- Positive potential: of all possible games with the current hand, all scenarios where the agent is behind but wins at the end.
- Negative potential: of all possible games with the current hand, all the scenarios where the agent is ahead but loses at the end.

The components of hand potential can be calculated as follows:

$$PPOT_n(h_i) = |\{\forall \mathrm{x} \in [\mathrm{D}\backslash \mathrm{h}_i]^5 \colon \forall \mathrm{y} \in [\mathrm{D}\backslash \mathrm{P}_i]^{\mathrm{nround}} \colon HS_n(h) \leq HS_n(x) \wedge HS_n\big(h + ж(\mathrm{y})\big) \geq HS_n(y) \wedge \mathrm{x} \supseteq \mathrm{S} \wedge \mathrm{y} \supseteq \mathrm{S}\}|$$

$$NPOT_n(h_i) = |\{\forall \mathrm{x} \in [\mathrm{D}\backslash \mathrm{h}_i]^5 \colon \forall \mathrm{y} \in [\mathrm{D}\backslash \mathrm{P}_i]^{\mathrm{nround}} \colon HS_n(h) > HS_n(x) \wedge HS_n\big(h + ж(\mathrm{y})\big) < HS_n(y) \wedge \mathrm{x} \supseteq \Omega \wedge \mathrm{y} \supseteq \mathrm{S}\}|$$

EQ12

given that nround is 2 when calculating for the Flop round and 1 when calculating for the Turn round. We also consider the function $ж\colon [\Delta]^{5..7} \rightarrow [\Delta]^{3..5}$ which extracts the community cards from any given hand of 5 to 7 cards. The main advantage

of this method is the consideration of Texas Hold'em upcoming rounds. In Figure 15, the average distribution of the PPOT and the NPOT components is shown, through heat maps. It is possible to perceive that there are higher concentrations of high PPOT values for closer cards (which are more likely to score a straight). As for the NPOT values, the hands with cards with lower ranks have a higher negative potential.

This presents the same result as Hand Strength in the River round (because the hand cannot evolve any further). Moreover, this method cannot be used in Pre Flop rounds, because it is not possible to calculate the hand strength for a two cards hand. This might be solved by combining this algorithm with Chen Formula (see at the end of this section). Similarly to the hand strength, if the Hand Potential is modified to only iterate over cards that the opponents might have [63], it is possible to obtain a better estimate of the winning ratio.

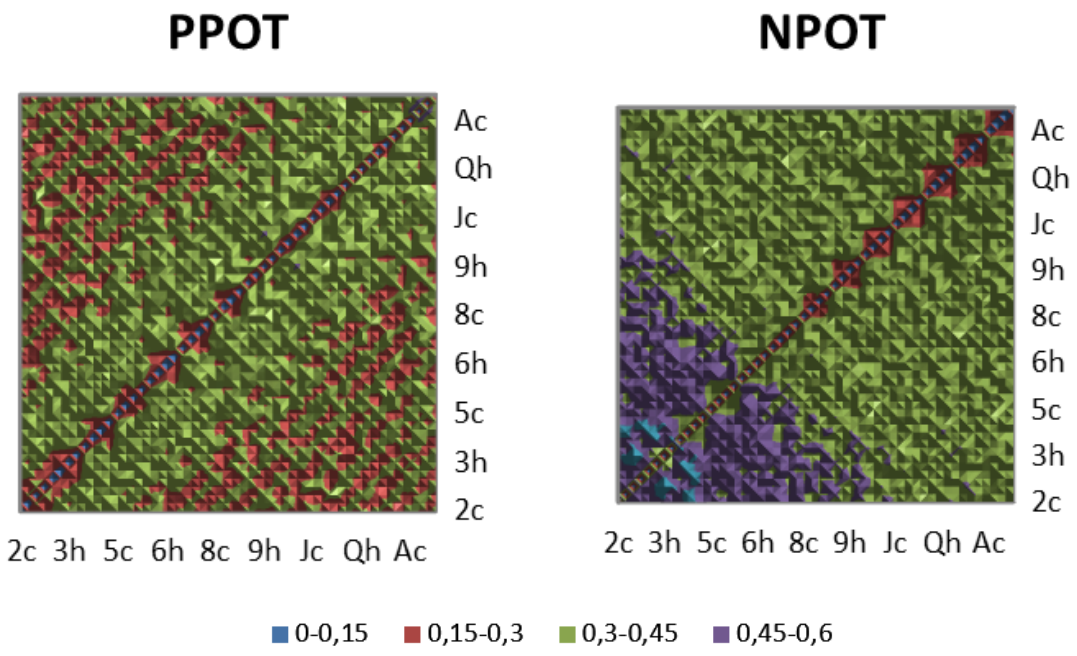


Figure 15 – Heats maps for PPOT and NPOT against 1 opponent.

3.5.3 Effective Hand Strength

The probability of winning can be calculated by combining the Hand Strength with the PPOT and NPOT components.

$$\mathrm{P}_n(\mathrm{win}) = \mathrm{HS}_n \times (1 - \mathrm{NPot}_n) + (1 - \mathrm{HS}_n) \times \mathrm{PPot}_n$$

EQ13

By setting the NPOT to 0, it is possible to determine the effective hand strength, which is the probability of the hand either being the best or improving to it.

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

EQ14

Through the observation of the Effective Hand Strength heat map (Figure 16), one can find that it has a similar structure to the simple Hand Strength map.

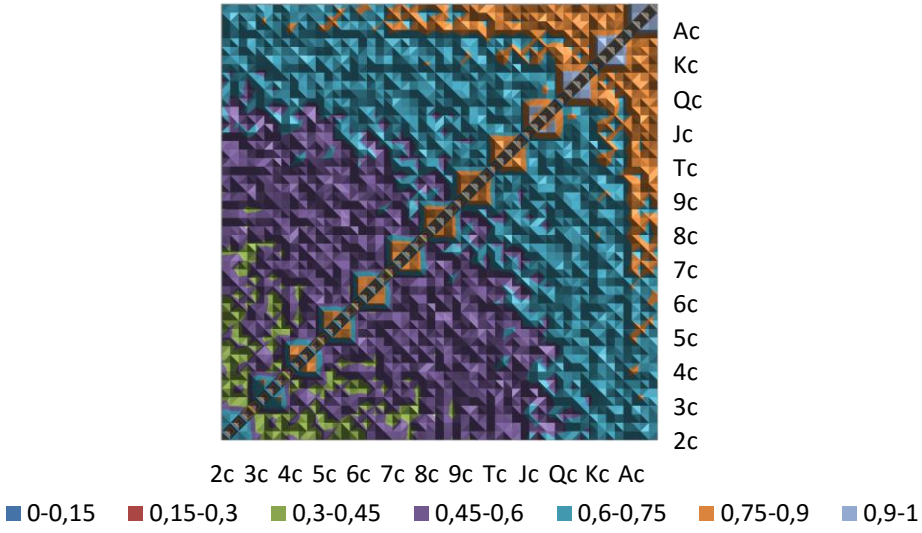


Figure 16 – Effective hand strength heat map against with 1 opponent.

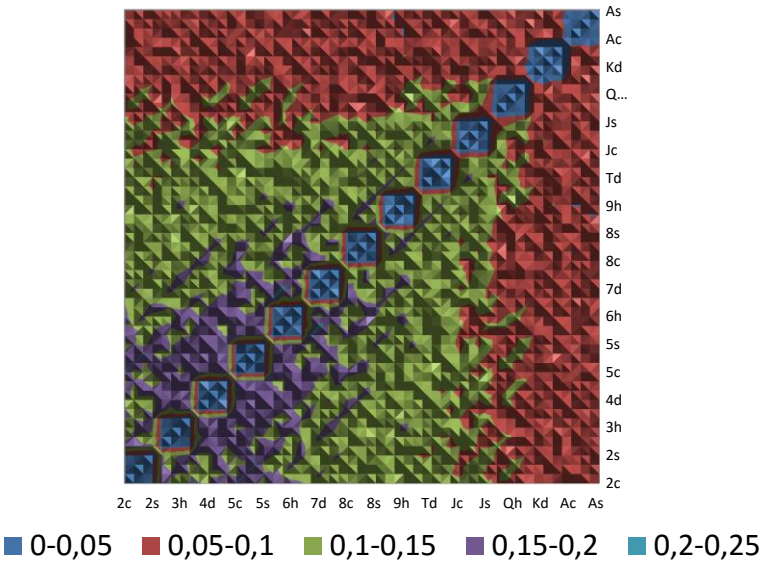


Figure 17 – Difference between effective hand strength and the hand strength.

For this reason, the additional computation time needed to calculate the effective hand strength might not be worth. To confirm this suspects, we computed the $EHS_1 - \mathrm{HS}_1$ heat map (Figure 17). In this map, it is possible to observe that the EHS method increases the value of all hands with special focus on hands with less hand strength. This happens because low scored hands have more potential to grow than "already made hands".

3.5.4 Incomplete Hands

In this section the Pre-Flop round of the game is addressed by showing how to compute odds when all community cards are still hidden.

One example is the Chen method. Chen is a fast naïve method developed by the professional poker player William Chen [64]. This can determine the relative value of the pocket hand. The main advantage of this over hand strength is that it does not need to generate permutations of card sets. For this reason, this algorithm is much faster than previously presented approaches.

```
function Chen(card1, card2){
     score = Max(Score(card1), Score(card2))
     if(card1.suit == card2.suit)
           score = score + 2
     switch(abs(card1.rank-card2.rank))
           case 0: score = score * 2
           case 1: score = score + 1
           case 2: score = score – 1
           case 3: score = score – 2
           case 4: score = score – 4
           default: score = score – 5
     return score
}
```

Figure 18 – Chen code implementation example

The algorithm is composed of two functions. The Score function returns a real number that scores a card (10 for Ace, 8 for King, 7 for Queen, 6 for Jack and rank / 2 for remaining). For instance, any ace card has the highest score possible (10). The Chen Formula function returns an integer which represents the value of the hand. Thus, the maximum returned value is 20 for a double Ace hand.

3.6 Opponent Modeling in Poker

Opponent modelling consists in classifying the opponents in order to make accurate predictions of their future actions in the game. In this section we present some techniques based on expert knowledge that are still the baseline approaches for abstracting used in very recent agents.

One of the most used player classification is the Sklansky classification, explained in [16]. The classification is based on statistical measures about the opponents and they do not usually consider the game results. The measures that can be used to classify players are:

- **Voluntary Put In Pot (VPIP) –** The percentage of how often a player puts money in the pot in Pre-Flop round by calling or raising. It does not count blind-bets.
- **Pre-Flop Raise (PFR)** - A percentage measure of how often a player raises pre-flop compared to calling or folding.
- **Aggression Factor (AF)** - The ratio between raises and call actions (Number of Raises / Number of Calls). Checks do not count.
- **Flop Continuation Bet (FCB)** – The percentage of times where a player makes more than one raise on the same Flop.
- **Fold Versus Flop Continuation Bet (FvFCB)** – The percentage of times that a player folds after doing two or more raises on the same Flop.

On the following tables (adapted from[22]) we present several player classifications based on these measures and ways to explore the opponents that classify on them.

[22] The explanation of Poker Statistics: http://pokerai.org/wiki/index.php/Player_statistics

Table 6 - Tight Aggressive Players.

Classification	Tight Aggressive
Conditions	VPIP: 12-16, PFR: 9-14, AF: >2.0
Description	This type of player will play a few times, but when he does he probably has a premium hand. When in game, this type of player plays very aggressively (high stakes).
Exploit	This type of players is difficult to exploit because they only play premium hands, despite being predictable. However, a good loose player cannot take advantage over tight aggressive players.

Table 7 – Nit/Rock players.

Classification	Nit/Rock
Conditions	VPIP: 7-11, PFR: 5-7
Description	This type is even stricter than the tight aggressive player. A rock player only plays a very small set of hands, usually above QQ in Pre-Flop and Pair after the Flop.
Exploit	Blind stealing. Since this type of players play fewer hands, any flat Call on the Pre-flop round should be answered with a Raise.

Table 8 – Loose Aggressive Players.

Classification	Loose Aggressive
Conditions	VPIP: 17-24, PFR: 15-22, AF: 2.0-5.0
Description	They play aggressively a wide range of hands. They usually use position and present good opponent modelling capabilities, being able to make great profit from getting their opponents to fold their better hands, using the concept of fold equity.
Exploit	Good Loose Aggressive players present few weaknesses. Strategies that could work against this type of opponents are Hammer or Rope-a-dope.

Table 9 – Manic/Aggro Donk players.

Classification	Maniac/Aggro Donk
Conditions	VPIP: 30-100, PFR: 30-60, AF >4.0
Description	This type of players bet and raises almost any hand and they rarely fold. In the long run, these players lose a lot of money.
Exploit	A good tight strategy works against these players.

Table 10 – Calling station player

Classification	Calling Station
Conditions	VPIP: 18-100, PFR: 0-15, AF: <2.0
Description	These players call almost every hand and they only raise (little) when they have a very strong hand.
Exploit	Tight strategy.

Table 11 – Short stacker player.

Classification	Short Stacker
Conditions	VPIP: 5-9, PFR: 4-9
Description	This type of player only applies in cash games. These players only enter the table with a 20 Big Blind stack and they either fold the hand or go All-In (with premium hands) at the Pre-Flop. They often play in position and only with hands above TT.
Exploit	These players are predictable. Blind stealing works well against this type of players.

Table 12 – Loose passive player.

Classification	Loose Passive
Conditions	VPIP: >30, PFR <15, AF <2.0
Description	Plays a wide range of hands passively. They are similar to calling stations but they fold more often.
Exploit	Tight and aggressive strategy.

3.7 Poker Books

There are many books that give advices on how to proficiently play Poker, written by known professional players or mathematicians. A Poker book normally gives a set of tips about game strategies and how these can be explored, showing common errors of each type of player and real examples of plays in important tournaments. Obviously, many of these tips are subjective and depend heavily on the game situation, so each player should always make his or her own strategy depending on its opponent's behaviour.

The next sub-sections briefly describe some of these well-known books. Many of the concepts present in these books have been constantly used in the development of the current state of the art approaches on computer Poker, especially for wisely abstracting the game information. The concepts in Section 3.6 were based mainly on the work of these authors.

3.7.1 The Theory of Poker

The Theory of Poker [16] is one of the most important books about Poker playing ever printed. It was written by the professional gambler David Sklansky and the first edition came out in 1987 (almost three decades ago!). Although being old, much of the content is still a reference for professional poker players and computer Poker researchers.

This book presents a complete overview of Poker theory in all main variants with some examples about each concept. Sklansky starts the book by explaining what he calls *"The Fundamental Theorem of Poker"*:

> *Every time you play a hand differently from the way you would have played it if you could see all your opponents' cards, they gain; and every time you play your hand the same way you would have played it if you could see all their cards, they lose. Conversely, every time opponents play their hands differently from the way they would have if they could see all your cards, you gain; and every time they play*

their hands the same way they would have played if they could see all your cards you lose.

David Sklansky [16]

In this theorem, Sklansky is clearly talking about Nash-equilibrium theory. When a player does not play its optimal strategy and deviates, he or she makes a mistake, therefore losing. For this reason, to win in Poker a player should take advantage of situations where the opponents do not use their equilibrium strategy.

The book also presents concepts about pot odds, the value of deception, getting and giving a free card, semi-bluff, raising correctly, check-raising, using position, bluffing and techniques for reading hands.

3.7.2 Hold'em Poker for Advanced Players

This [65] is another book written by David Sklansky (and Mason Malmuth) with the first edition coming out in 1988. This book is considered the continuation of *The Theory of Poker* but focused on the Texas Hold'em variant of Poker.

This book introduces the Sklansky groups of cards in Texas Hold'em. There are 169 distinct sets of two card starting hands in Texas Hold'em. In this book, the authors divided those sets into eight different groups (see Table 13), according to strength and playability. Each group has a description of how to play with those cards.

Table 13 – Sklansky and Malmuth groups

Group	Hands
1	AA, AKs, KK, QQ, JJ
2	AK, AQs, AJs, KQs, TT
3	AQ, ATs, KJs, QJs, JTs, 99
4	AJ, KQ, KTs, QTs, J9s, T9s, 98s, 88
5	A9s...A2s, KJ, QJ, JT, Q9s, T8s, 97s, 87s, 77, 76s, 66
6	AT, KT, QT, J8s, 86s, 75s, 65s, 55, 54s
7	K9s...K2s, Q8s, J9, T9, T7s, 98, 64s, 53s, 44, 43s, 33, 22
8	A9, K9, Q9, J8, J7s, T8, 96s, 87, 85s, 76, 74s, 65, 54, 42s, 32s

It is possible to easily compute this table using the professional player Bill Chen's formula (see 3.5.4). This formula is presented in his book "The Mathematics of Poker" [64] and it calculates the relative value of the pocket cards.

3.7.3 Super/System: A Course in Power Poker

This book [66] written by Doyle Brunson and other known professional Poker players was first published in 1978. This was the book that presented some of the most important concepts about strategy in Poker for the first time, even before Sklansky's publications.

One important characteristic of this book is that the author defends his own aggressive playing style over passive playing styles. Solid theoretical proof now proves that aggressive play is usually superior to a more conservative style, but until the date of publication of this book, there were no previous studies about it [25].

3.7.4 Gambling Theory and Other Topics

This book [67] gives a understanding of how gambling works generally, by explaining the fluctuation that create illusions among the players about how strong they are in the game. It focuses particularly on the mathematics of poker, and how a player can take advantage of them to win.

3.7.5 Every Hand Revealed

Every Hand Revealed [68] is a more recent book written by Gus Hansen (known by Poker players as "the madman"). This book presents a very practical approach for learning Poker, where the author shows all the hands that he played during Aussie Millions Poker Tournament in 2007 – one of the most important Poker tournaments that he won against 746 competitors. For each hand, Hansen explains the logic behind his decision usually with mathematical support.

Since he is a Loose-Aggressive player, he plays differently than most of the best professional Poker players. For this reason, this book presents an interesting approach for a different winning strategy.

3.8 Poker Simulators

A Poker Simulator is a software tool that allows for Computer Poker researchers to test their agents against other agents or human players, allowing them to predict the agents' success at long term, before putting them in a real life environment. In the case of Poker this is especially important, because assessing agents in real-life environments can cost a lot of money.

3.8.1 LIACC's Texas Hold'em Simulator

LIACC Texas Hold'em Simulator (Figure 19) is a software capable of simulating Limit or No-Limit Texas Hold'em games. It has a client-server architecture where the server communicates with clients (agents) through sockets with a predefined TCP communication protocol. The software was developed in C/C++ [63].

This simulation software supports up to 10 players which could be either human or automated agents (it provides a client for games with human players). The communication protocol between clients and server is based on the ACPC competition protocol (see Section 3.8.3), so the developed agents are ready to compete there.

Before starting the simulation, some game options can be defined, such as chip stacks, blind value, log file name, etc. The created log file stores information about bets and how much money each player wins/loses in each game.

Figure 19 – LIACC's Texas Hold'em Simulator.

3.8.2 Poker Academy

One of the best resources for testing a Poker agent is the simulation software named Poker Academy[23] (Figure 20). In this simulator it is possible to compete against the best agents developed by CPRG at the University of Alberta. It was launched in December 2003 as a tool for professional player training.

Poker Academy provides a Java based API (named the Meerkat API) that allows Computer Poker researchers to plug in their own custom bots. This gives an excellent environment for bot development as it is possible to easily put a custom bot playing against the best bots developed until now, in a quality GUI. The program also keeps track of all the hands played and can display comprehensive charts and analysis of the player statistics over time.

One of the problems of Poker Academy is that it is misfit for extensive simulations, because of the heavy user interface that results in low simulation speeds. Another problem is that it is not possible to start a simulation without a human player, which means that in each simulation there will always be an additional ghost player that the user must configure to always fold its hands, adulterating for this reason the simulation results.

The project was discontinued because it failed commercially, but then got back in 2015 as a new training tool called Poker Genius[24].

Figure 20 – Poker Academy.

[23] Official website: http://www.poker-academy.com/
[24] Poker Genius Official Website: http://poker-genius.com/

3.8.3 ACPC Poker Server

The ACPC Poker Server (Figure 22) is the application built for the AAAI Annual Computer Poker Competition. It is known for being fast, by simulating thousands of games between poker agents in milliseconds (with a personalized timeout that kicks an agent out if it is very slow). It is composed by three applications:

- **Client** – a sample client (in C) that is provided with the software package that gives a good starting point for personalizing agents.
- **Server** – it runs the game and deals cards for the several connected agents. Its architecture and its communication module (simplistic protocol over TCP) allows for any agent written in any language to connect to it.
- **Observer** – it is possible to implement observer applications that can watch the match (however without knowing the hidden cards of each player).

This simulator is very simple but it lacks an easy to use API (all it has is a library written in C that is efficient but very hard to use). It has some personalization options. See the configuration file example in Figure 21.

```
GAMEDEF
limit
numPlayers = 3
numRounds = 1
blind = 1 1 1
raiseSize = 1
firstPlayer = 1
maxRaises = 1
numSuits = 1
numRanks = 4
numHoleCards = 1
numBoardCards = 0
END GAMEDEF
```

Figure 21 – ACPC Poker Server – server configuration

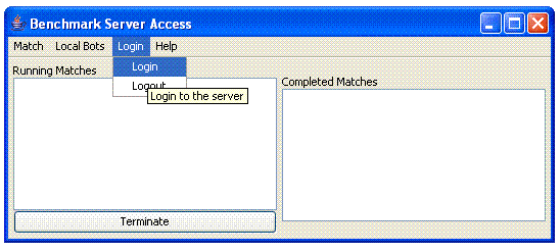


Figure 22 – ACPC Poker Server – User interface.

3.8.4 Open Meerkat Poker Test bed

Open Meerkat Poker Test bed[25] (see Figure 23) is an open source implementation of the Meerkat API for running Poker games. It imitates the Poker Academy simulator; however it is much faster because it lacks a heavy user interface.

This application supports Fixed/No-Limit cash games with automatic rebuy. It generates bankroll evolution plots, implements seat permutation to reduce game variance (replay games with same cards but with different seat order) and generate game logs. It also shows an online bankroll evolution chart. The main issue of this application is that it still has some bugs in the game's algorithm, only supports even number of players and it does not have built-in playing agents (only dummy agents with very basic strategies).

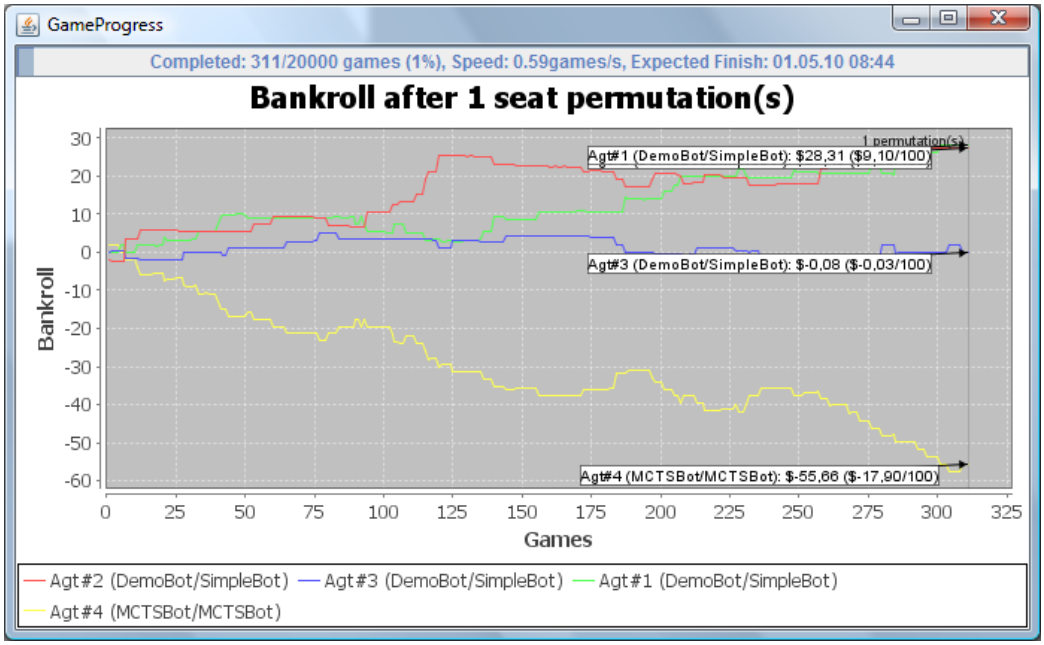


Figure 23 – Open Meerkat Poker Test bed.

3.8.5 IRC Poker Server

Even before the real-money online poker sites were popular, several human players played with text based scripts for the Internet Relay Chat protocol. At the same time, the first agents appeared, which allowed the first matches (without real money) of Texas Hold'em Poker. Many versions of the first rule-based Poker agents (see Section 3.2.1) such as Loki were assessed in matches on those servers. Despite the fact that

[25] Available at https://code.google.com/p/opentestbed/

matches didn't use real-money, the players needed to raise their skill level in the system to be entitled to play against stronger opponents, which increased the challenge factor on those matches. Some software agents were relatively successful there but notwithstanding the levelling system, the players never play the same way as if they were betting real money (we also do not know how skilled were the players there). Both the software and several gigabytes of match logs are still freely available for download.

3.9 Interaction between Poker Agents and Human Players

Testing Poker agents in simulated environments is very important and can give empirical proof of the agent's potential and theoretical success. Nevertheless, without testing agents in a "real life" environment against human players, their skills can never be properly validated. There are some tools that provide this type of interaction between Poker agents and human players, which will now be presented.

This kind of applications are called **Poker bots**. The tools presented in Sections 3.9.1 and 3.9.2 are already deprecated, since their compatibility with the most important Poker rooms is non-existing or with a lot of bugs – however they still work on less popular rooms.

The creation of an application that plays Poker in rooms is very complex task because the online casinos usually do not provide an API for it – it requires an application that interacts with the casino client through image processing. Moreover, these applications require constant updates, because the casino client applications are always changing. For this reason there are not much public software that supports these functionalities. There is also no research on how to construct this type of applications, to the best of the author's knowledge.

3.9.1 WinHoldEm

WinHoldEm[26] is a commercial programmable poker bot (see Figure 24) that allows for users to connect their agents to real money tables in online casinos. Users that have

[26] Available online at http://www.winholdem.net/

software development skills can develop and compile their own bots in standard C/C++.

This type of application is often not permitted in Poker tables, and its usage could result in banning. However, WinHoldEm uses an advanced stealth module to avoid detection, and it can be used on some important sites like PartyPoker[27] without significant problems. This tool also allows for automated collusion (teamwork between players to get an unfair advantage).

Despite not being permitted, this type of applications may present the only way to test the agent in a real-life environment. There are also some Poker sites that do not officially support the use of these tools but they allow them (if no collusion happens between bots).

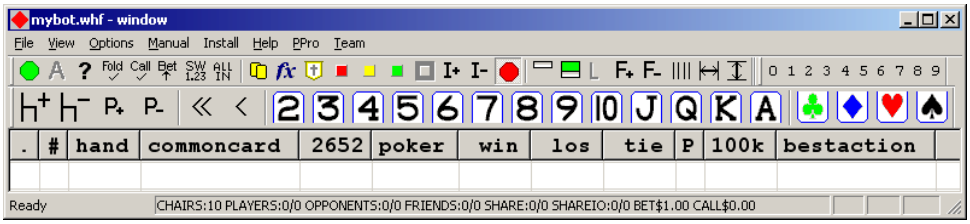


Figure 24 – WinHoldEm graphical interface

3.9.2 OpenHoldEm

OpenHoldEm[28] is an open source screen scraping framework and programmable logic engine for the online Texas Hold'em poker game. This framework has similar functionalities to WinHoldEm but it has the advantage of being free (it is usually known as a free WinHoldEm). Unlike WinHoldEm, this tool does not support automated collusion. This tool's main components are:

- A parameter driven engine for screen scraping and interpreting game states (Table Maps)
- A logic engine for making poker decisions based on the game states
- A simplistic scripting language for describing how these poker decisions should be made (using the Spirit parser library)

[27] Official website: https://www.partypoker.com
[28] https://code.google.com/p/openholdembot/

- Various interface mechanisms allowing for the creation of decision logic via alternative means (C++, Perl, etc.)
- An engine for applying the poker decision to the casino table (Auto player)

This open-source project was recently archived, however there are video reports (on YouTube) from 2014 / 2015 claiming that it still works.

3.10 Emotions in Poker

Affective computing is a research domain whose goal is to attempt to create systems that can recognize, interpret, process, and or simulate human emotions. It is an interdisciplinary field spanning from computer science and psychology [69].

In competitive games, if an artificial player is capable of interpreting the emotional state of human opponents, it can adapt its strategy, giving a more appropriate response: e.g. being more aggressive when the opponent is in a depressive state or being more conservative when the opponent seems more serious. Since the machines currently do not possess emotions, except when simulated, they also do not have the disadvantages associated with them during a game, i.e., they do not get tired, frustrated, anxious, etc. Thus, a match between a human player and an agent of similar skill level would often result in victory for the agent because it is not affected by any emotional state, thus being able to keep its strategy and make no mistakes. A very complete review about this issue can be found in [69].

One important concept is tilt. Tilt is an emotional state in a game of Poker, based on emotional confusion or frustration that affects the player's behaviour in the game, which causes the player to use a less optimal strategy than usual.

This concept is defined and explored in [70]. The authors state that all gamblers experience tilt, and their reactions to tilt and to tilt-inducing situations partly determine whether or not gambling becomes a major problem.

Generally tilt is experienced by big losses of money in Poker. However, not only big losses bring a deviation from the original optimal strategy. Big gains can also affect

the strategy of a human player because they might stimulate overconfidence, which can result in careless play [71].

3.11 Summary

In this chapter the current most relevant methodologies for the creation of Poker agents were presented and discussed. Some tools to support agent's development were also presented as well as some expert knowledge notions and information sources. This chapter serves as support for the rest of the document and is referenced throughout the document when necessary.

Chapter 4

Simulation and Tools

This chapter describes the general architecture of the simulation systems and tools developed to support Computer Poker research. Three different systems were developed: one simulation system that is used to test Poker agents before entering competitions or playing online, which provides several features that facilitate their assessment; a Poker description language which allows for describing sets of rules to define customized Poker games – allowing future contributions in the domain of general game playing and finally a Poker Bot system which allows for software agents to play online without human players knowing that they are actually playing against an agent, which reduces the psychological impact on humans that participated on some of the tests described in Chapter 7.

4.1 LIACC Poker Simulator

New Computer Poker developments are usually made through the implementation of software agents. A Poker agent is software that replaces a human in the task of playing Poker, by taking decisions without any human intervention. Since playing Poker can be considered a repetitive task for a human player, the development of software agents not only allows progresses in computer science (as explained in Chapter 2) but also has potential commercial value to professional Poker players. If they were able to create agents in their image, they could be rewarded for their effective know-how of the

game and not by their physical endurance or patience. This is true, because most lucrative players are usually the ones that play more carefully and more games.

The challenge in developing agents for incomplete information games resides in the fact that the decision that gives maximum utility for a given information set is not always ascertainable. In light of this, simulation systems are indispensable for accurate assessment of agents' capabilities. Nevertheless, and as reviewed in Section 3.8, current systems do not accommodate the needs of computer poker research since they were strictly designed as training tools for human players to improve their skills (with the exception of the ACPC simulator). In order to contribute towards the improvement of computer poker research, a new version of the LIACC Poker Simulator was developed from scratch (it was not based on the one described in Section 3.8.1).

This simulator considers scientifically unexplored game modes with the purpose of providing a more realistic simulation environment, where the agent must play carefully to manage its initial resources – the environment follows more closely what actually happens in online rooms at popular casinos. Several other features were introduced in the simulated environment to speed-up the simulator, namely the inclusion of table seat permutation [28], which reduces the variance of the results, requiring therefore much less iterations to properly validate an agent.

An evolutionary simulation feature was also included so as to provide support for the improvement of adaptive strategies. The simulator has built-in odds calculation, an agent development API, other platform agents and several variants support and an agent classifier with realistic game indicators including exploitability estimation.

Another important aspect of the new system is the consideration of bankroll[29] management – a key concept considered essential by professionals for proper game play. The importance of bankroll management can be explained by the gamblers ruin theorem [72]. This theorem states that even if players use a strategy that has positive expected value[30], they will still be very likely to be bankrupt if they raise the stakes[31] when they win but do not lower them when they lose.

[29] Bankroll – amount of money that a given player reserved for playing Poker.
[30] Expected value – average amount of money won per play.

4.1.1 Goals

In order to overcome the limitations found in previously developed Poker simulators, a new simulator has been created which aims to integrate the most important features present in other simulators with new features that will certainly lead Computer Poker research into new directions. The requirements of the new simulation system are:

- An expandable architecture to support the creation of agents or the introduction of game variants. This includes an agent development API.
- New game modes such as ring, which allow researchers to explore the paradigm of bankroll management.
- Evolutionary simulation of Poker games, which encourages studies about strategy evolution through the principle of natural selection. This feature is not known to be natively supported by any Poker simulator.
- A set of validation tools that allow for a quick and precise assessment of the agent capabilities to predict their performance in different real-life like environments.

4.1.2 Agent Modelling

The simulation system described in this thesis uses a multi-agent architecture where an agent represents a Poker player. Many types of agents were created for this simulation platform, each one of them represented in the code by a class. The way each class relates to others is depicted in the UML class diagram of Figure 25.

Poker Agent – it is an abstract class based on the Meerkat API [73] that represents any agent in the system. The class contains a set of abstract methods that represent the events that each agent has to answer to during the simulation. Thus, to create an agent that works in this system it is necessary to extend this class. Agents must implement a set of methods corresponding to events of the game:

- *pocketCards(Card[], Seat)* – occurs when the agent receives its pocket cards.

[31] Stake – amount of betted money per game.

- *observeAction(GameInfo)* – the main routine of the agent. It is called when the agent is requested to perform an action.
- *actionEvent(Seat, Action)* – A player in a given seat has performed an action.
- *winEvent(Seat, Amount, Card[])* – A player in a given seat has won an amount of chips with a given hand.
- *showdownEvent(Seat, Card[])* – player in a given seat has shown his cards.
- *gameOverEvent()* – the current game is now over.

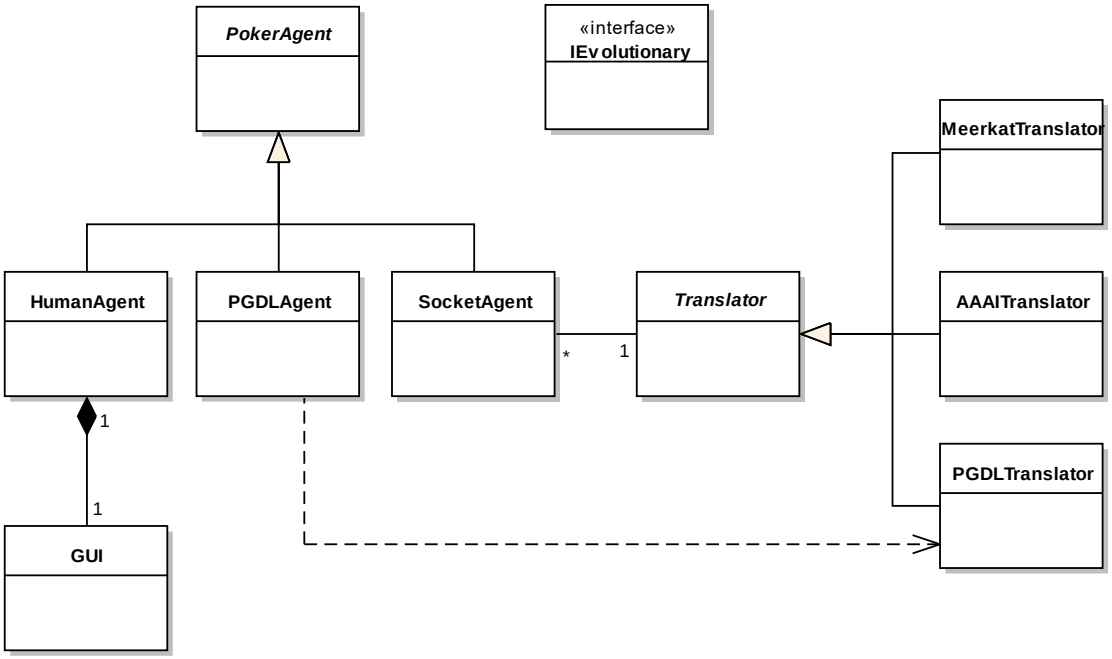


Figure 25 – Poker Agents class model.

HumanAgent – this agent extends the class PokerAgent and redirects the game events to a graphical user interface (GUI). This GUI is controlled by a human player. Thus, this class represents a form of interaction between human and artificial players.

PGDLAgent – this agent extends the class PokerAgent and allows for integrating agents developed with the PGDL System. This agent requires the **PGDL Translator** which is a parser for PGDL documents. The specifications of the PGDL System are described in Section 4.2.

SocketAgent – the socket agent is responsible for communicating with external agents developed for other simulation platforms. This way, any external agent from Poker Academy [73] or AAAI Server [28] can be used in this simulator without need of recoding, using the new *PokerAgent* class. The communication process is demonstrated in Figure 26. When a *SocketAgent* receives a request, it chooses the correct translator and then sends a translated request via sockets to an external application that is linked to the external agent. The external agent then sends the response all the way back to the *SocketAgent* and then the *SocketAgent* plays accordingly.

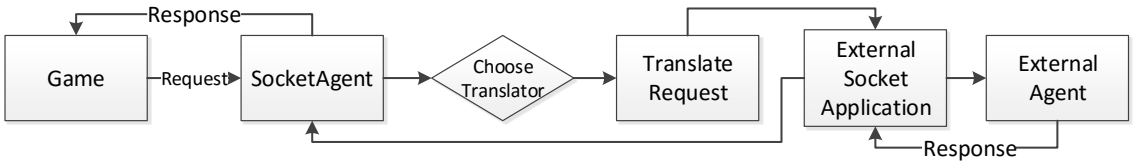


Figure 26 – Communication between the Socket Agent and the External agent.

IEvolutionary – this optional interface adds three methods to any class that extends from *PokerAgent*. These methods allow the agent to participate in evolutionary simulations. The methods of this interface are the following:

- *ReproduceAsexually* – this method should return a new child agent created by the current one, with upgraded parent features;
- *ReproduceSexually* – this method should return a new agent created by crossing characteristics from both this agent and another one;
- *Fitness* – this method returns a number that measures the level of adaptation of the agent to the current environment. The fitness could be for instance the average expected value against all opponents.

4.1.3 Simulation System Architecture

The architecture of the simulator is depicted in Figure 27. The simulator was implemented in JAVA to maximize the compatibility with several systems. The simulator is composed of the following components:

- Hand Rank Server – a server that is used to calculate the rank of the Poker hands based on the algorithms described in Chapter 5;
- Simulation Server/Poker Simulation Library – the application that is responsible for simulating Poker games;
- Logging database – all agent moves are registered in a database for future profiling and result analysis;
- Poker Agent – this entity represents an abstract Poker agent;
- Poker Library – definition of general Poker data structures;
- Poker Statistics Library – calculates statistical indicators and thus validates agents;
- Poker GUI – user-friendly GUI to allow humans to play against the agents.

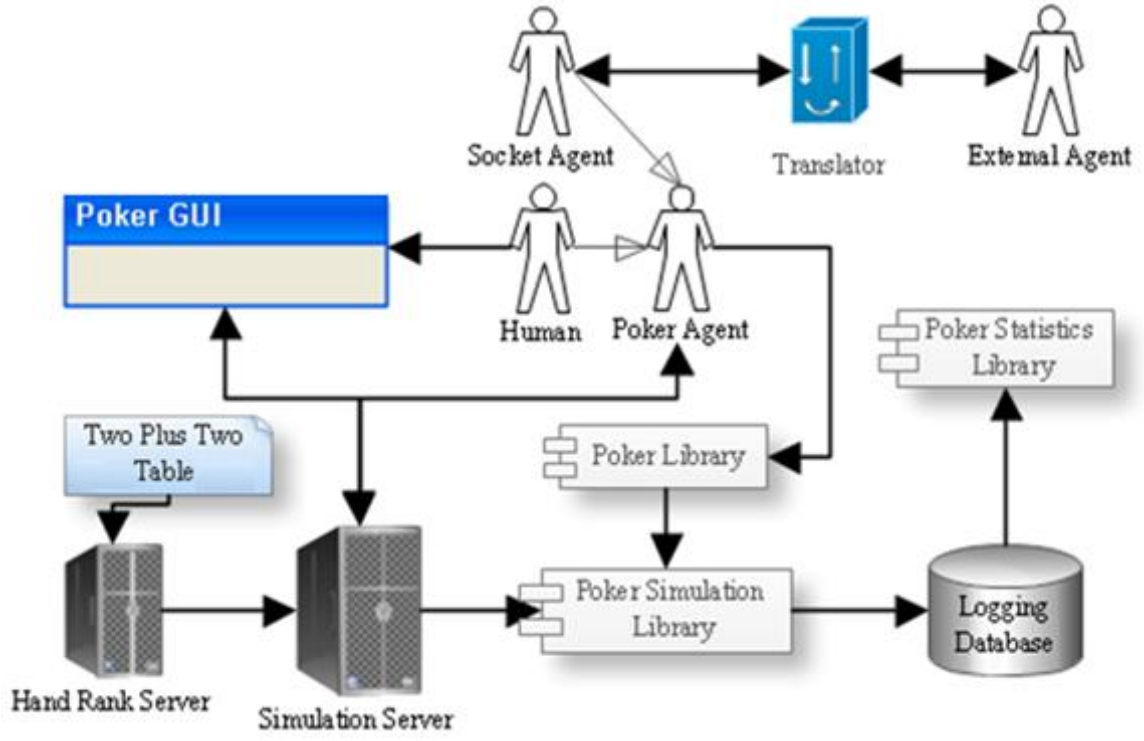


Figure 27 – Poker Simulation System Architecture.

4.1.3.1 Hand Rank Server

The hand rank server is a process that runs concurrently with the simulation server and that evaluates Poker hands for all agents. This was created to save memory since the fastest hand evaluating algorithm – TwoPlusTwo Evaluator [74] – must load a 130 MB

table. If each agent were to load the table individually it would be problematic in terms of memory usage, especially in the evolutionary simulation module where thousands of agents might be needed.

The hand ranking server uses a simple TCP communication protocol to provide different measures that evaluate the chance of winning: hand rank; hand strength; hand potential; effective hand strength and Chen formula. Table 14 presents the commands that can be sent to the server (<Hand> is a string composed of 5 to 7 cards like 'AsAd7s4d2c'). Already computed results can be optionally saved by the hand ranking server in a private database in order to speed up future requests. Pre-computed results consider hand isomorphisms, since that for instance, asking the Hand Strength for A♣A♥ is the same as asking for A♦A♠.

Table 14 – Hand Ranking Server Commands.

Command	Description
RANK <Hand>	Retrieves the rank of the hand.
HS <Hand> <NO>	Retrieves the hand's strength. <NO> = remaining adversaries.
HP <Hand> <NO>	Retrieves the hand's potential.
EHS <Hand> <NO>	Retrieves the effective hand strength.
CHEN <Card> <Card>	Retrieves the relative value of a hand with 2 cards.
ARS <Hand> <NO>	Retrieves the effective hand strength approximated using the Average Rank Strength tables (see Chapter 5)

4.1.3.2 Logging Database

The simulator has a database that contains records of all moves made by registered players, if the logging option is set. Figure 28 presents the class model of the database that was subsequently converted to a relational database model.

The database uses a data warehouse model which will help researchers to process the raw data. This produces some intentional redundancy in the data, namely the link between the Player and the Game classes that can be used to facilitate game analysis, reporting and data mining. The model is composed by the following classes:

— **Action** – represents an action in a given game performed by a player. This class represents the star table and thereby a key aspect of the simulator database. An action presents the full state of the game table when it took place, instead of only containing the action type and the value;

— **Game** – represents a game which is a set of actions;

— **Player** – represents a registered player in the game;

— **Simulation** – represents a simulation run on a date and time. It is a set of consecutive games;

— **Room** – some simulation modes require the concept of room/table i.e. the occurrence of games in parallel in the same simulation.

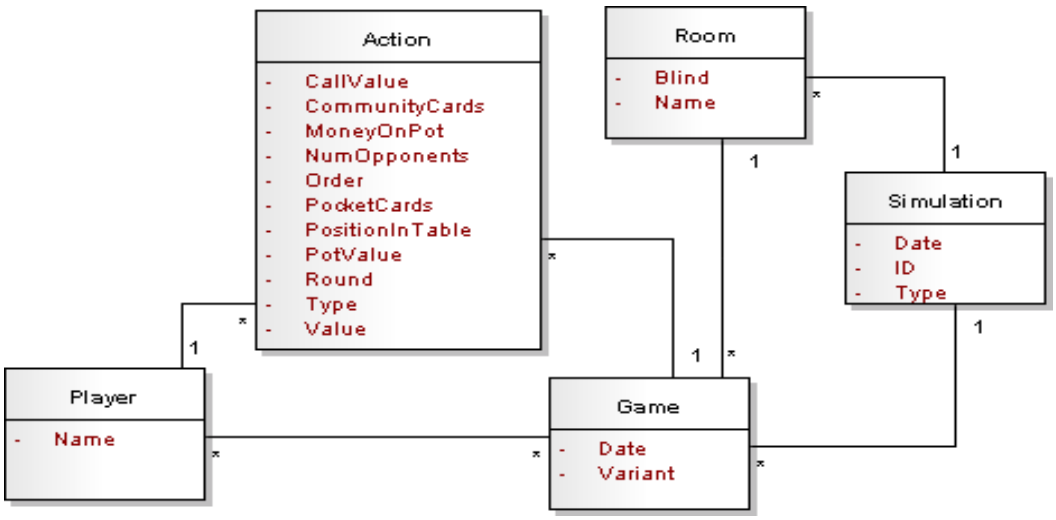


Figure 28 – Game moves database class model.

The used format is also helpful for case based reasoning agents, because of the presence of redundancy in the action table that aids the computation of approximate information sets [75].

4.1.3.3 Poker Simulation Module (Poker Simulation Library)

This module is responsible for performing the simulation itself. When the simulation starts the user will be asked which players will be part of the game, which simulation

mode to use and which Poker rules. The class diagram in Figure 29 shows the entity structure of the simulation module. The existence of simulation modes is one of the innovative aspects of the system and five different modes were considered.

- **Simple Tournament** – a simple tournament is a set of games that only ends when only one player remains. This kind of simulation allows testing the capabilities of the agent to manage its cash and the blind increase in order to win the tournament and avoid the gamblers ruin theorem [72].
- **Full Tournament** – this mode is similar to a simple tournament but with several gaming tables.
- **Cash Games** – the common type of simulation that is used to validate Poker Agents. It consists of a finite set of games with static blinds and player money reset at the beginning of each game. To reduce the variance of the results, table seat permutations is used – for each game positions are switch and the same cards are dealt, so everyone has equal chances. This type of simulation allows players to be tested on the long run, always on equal footing.
- **Ring Games** – this mode is similar to what happens in online casinos. The agent starts with a given amount of chips and must manage it in order to survive. In addition, the agent should choose the table that contains opponents that are more susceptible to its strategy and tables with blinds that do not present a risk of quickly losing all cash.
- **Evolutionary Cash Games** – this mode is similar to cash games simulation. However, in this mode, from time to time, natural selection is applied. This means that the agents with less fitness will be discarded and the other agents will reproduce, creating child agents that contain characteristics of both parents.

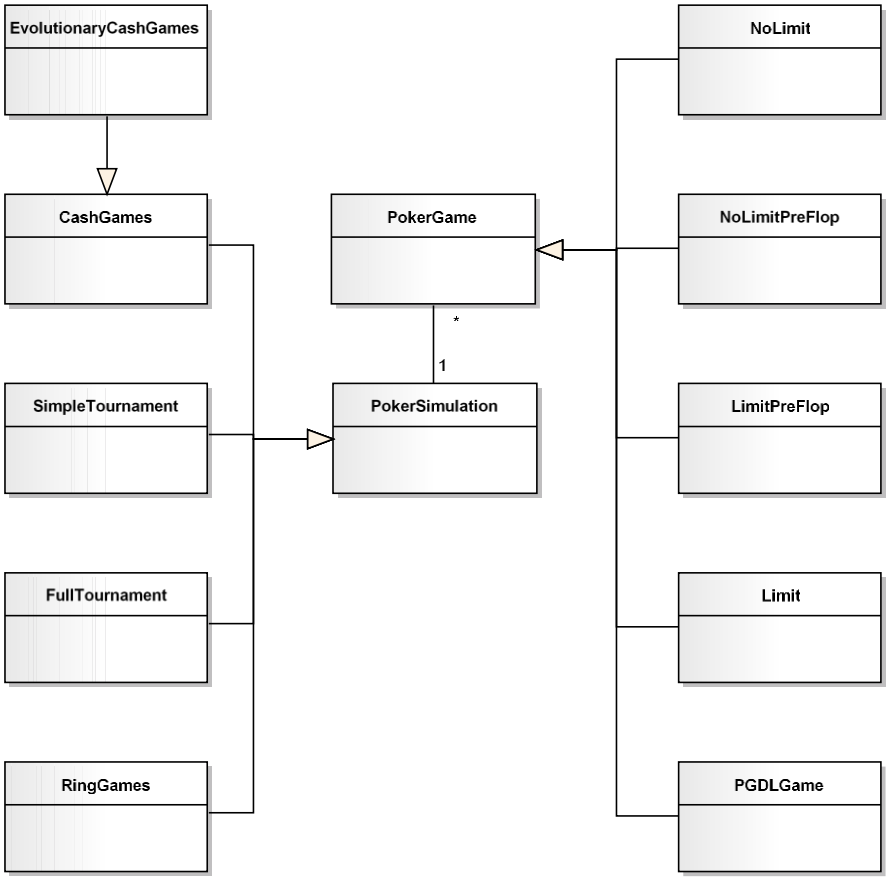


Figure 29 – Poker simulation module.

There are four main game types: Limit Texas Hold'em, No Limit Texas Hold'em, Limit Texas Hold'em Only Pre-Flop and No Limit Texas Hold'em Only Pre-Flop. The innovative part of the game types is the presence of "Only Pre-Flop" variants. These are variants of Texas Hold'em that only last until the Flop round therefore they do not have community cards. This variant is popular among new Poker researchers, given the much lower number of information sets than in full Texas Hold'em resulting in less abstraction for strategy computation. This system can be easily expanded by inheriting from the PokerGame class or by implementing game rules using the PGDL system (described on 4.2).

For the same reason, the variant Kuhn Poker was also included (as a particular implementation of a PGDL Game, later described in this Chapter). Kuhn Poker is a variant that only uses 3 to 13 cards (the number of card is a simulation parameter) and no community cards, resulting in a maximum number of 52 possible information sets. This allows researchers to quickly validate new approaches, with much less effort and

computation time needed, especially when working with algorithms such as CFR that can take weeks to finish for full Texas Hold'em.

4.1.3.4 User interface

In order to quickly configure the simulation parameters, a configuration GUI was developed (Figure 30). This GUI includes an optional and minimalist 2D visualizer (Figure 31) to observe the agents in action.

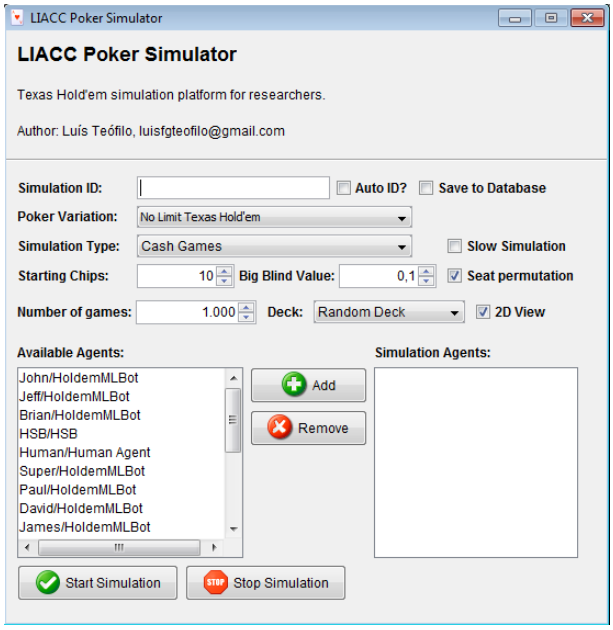


Figure 30 - LIACC Poker Simulator

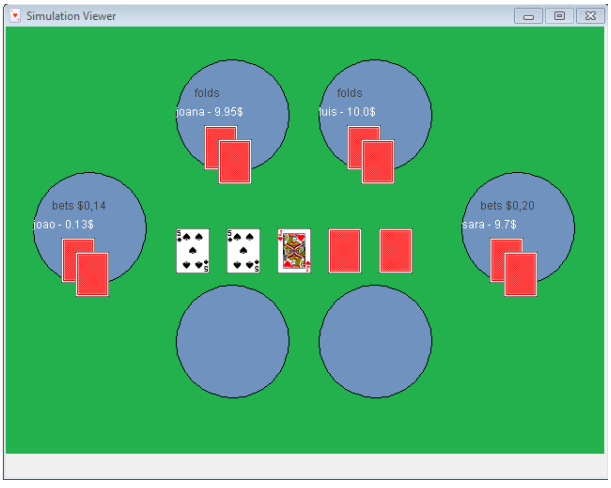


Figure 31 –LIACC Poker Simulator 2D visualizer

4.1.3.5 Evolutionary Simulation Model

The evolutionary model follows as the diagram in Figure 32. The simulation can be started by selecting the evolutionary parameters (number of iterations, population size: M, percentage of agents eliminated per iteration: n) and the agents that take part in the simulation. The population size is maintained throughout the simulation but it is renewed on every iteration. The simulation ends after a defined number of iterations.

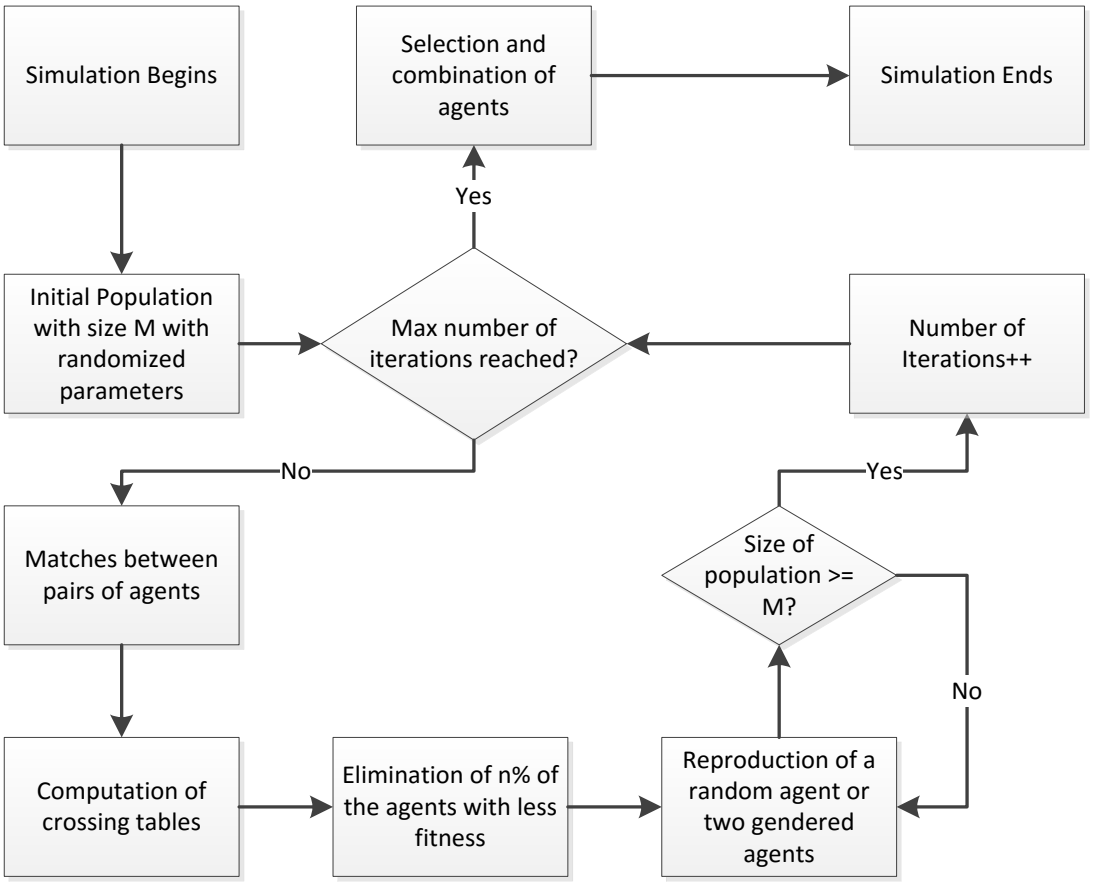


Figure 32 – Evolutionary simulation module.

4.1.4 Agent Assessment

After performing the simulations, the statistics module can be used to analyse the results. Three types of statistics were included:

Bankroll evolution – the evolution of the player cash during the simulation. This statistic shows the evolution of the agents' profit during a simulation.

Player indicators evolution – several indicators used by Poker experts are available in evolution plots and described in Table 15.

Exploitability analysis – the exploitability is the agent's utility against a best response agent. A best response agent is the average best possible strategy against one's own strategy. Calculating a best response can be done using CFR. Since Poker is a very large game, abstraction is needed to perform this operation in a timely manner.

This simulator provides exploitability computation by following the next steps:

- Selection of the level of card abstraction (0 to 100). The results are more accurate for lower levels of abstraction.
- Selection of the level of action sequence abstraction (0 to 100).
- Selection of the number of iterations for CFR and for final simulation.
- Computation of the best response strategy using the CFR algorithm with a desired level of abstraction;
- Final simulation and computation of the exploitability level.

Table 15 – Player statistical indicators.

Indicator	Description	Round
VPIP	Percentage of games where the player puts money in the pot.	Pre-Flop
PFR	Number of Raises / (Number of Calls + Number of Folds)	Pre-Flop
AF	Number of Raises / Number of Calls	Flop

4.1.5 Tests and simulator evaluation

The developed simulator was tested against other simulators in the matters of speed and features.

4.1.5.1 Benchmark Tests

In order to compare the speed of this simulator against previously developed simulators, a benchmark test was performed. The test consisted of repeating for 1.000 tries a simulation of 100.000 cash games, with 4 players without table permutation (since Poker Academy does not support it). The results are shown in Table 16.

As can be observed, the new LIACC simulator is the fastest one. The results were very close to the Open Meerkat Testbed, however the Poker Academy simulator was

much slower. This was due to the heavy user interface present in the Poker Academy software that slowed down the simulation process.

Table 16 – Simulator benchmark test results for 1.000 tries with 100.000 games and four players.

Simulator	Average Time (seconds)	Std. Deviation (seconds)
Open Meerkat Test Bed [76]	43.0	6.3
Poker Academy [73]	660.3	48.7
LIACC Simulator	27.7	1.8

Table 17 – Poker Simulators Comparison table.

Feature	LIACC Simulator	Open Meerkat	Poker Academy	Is Key Feature?
2D visualizer	Yes, Simple	No	Yes	No
Agent Development API	Yes	Yes	Yes	Yes
Bankroll Analysis	Simple	Simple	Complete	Yes
Card Rank Computation	Yes	No	Partially	Yes
Database support	Yes	No	?	No
Evolutionary Simulation	Yes	No	No	Yes
Expansible Architecture	Yes	Yes	No	Yes
Exploitability	Yes	No	No	Yes
Human players	Yes	No	Yes	No
Logging	Yes	Yes	Yes	Yes
Online play	No	No	Yes	No
Pre-developed agents	No (but PGDL)	Yes, Simple	Yes	Yes
Simulation Speed	Fast	Fast	Slow	Yes
Table seat permutation	Yes	Yes	No	Yes
Former agent support	Yes	No	No	No

4.1.5.2 Qualitative Comparison

Table 17 summarizes the comparison between the main Poker simulators. The first column presents the feature. The 3 subsequence columns present the main available simulators. The last column indicates if the feature is considered to be essential (in the author's opinion) for implementing a simulator that is fully suited for Computer Poker research.

The only missing features in LIACC's simulator are online play and pre-developed agents. Despite this simulator not providing pre-developed agents, this can be balanced by the *Former Agent Support feature* which allows the use of agents developed for other platforms. Moreover, the PGDL pre-built agents might also be used because this simulator is compatible with the PGDL system.

4.1.6 Summary

The new LIACC Poker simulator is scalable, fast and is able to lead Computer Poker research to unexplored paths. The key features of this system are the possibility of performing evolutionary simulations, tournament simulation and support for external agents. Also, this simulation system provides access to an extensive database that could be easily used for data-mining and better opponent modelling profiling in the future. Moreover, there could be significant improvement of agents' performance in real-life environments by analysing the comprehensive statistical indicators generated by the system.

This simulator is in final stages of development, with some extensive testing already done. Performance tests demonstrated that this simulator is faster than all the others it was compared with. The qualitative analysis also shows that this simulator outperforms previously developed simulators in terms of research aiding features and proper agent assessment.

4.2 Poker Game Description Language (PGDL)

The term Poker is commonly wrongly recognized as a game. Poker is actuality a category of games with hundreds of different variants, which differ from each other by

their betting structure, the number of cards in the deck, the way the winner is determined, among other rules. These features represent by themselves unique challenges in Poker agent development.

However, to the best of the author's knowledge there is not a single unified description model that allows for game playing agents to be tested across different Poker variants inexpensively. This is rather important when developing new Poker playing agents for two main reasons:

- Each poker variant has unique characteristics in its rules that assess different components of the agents' strategies. If one develops an agent under a representative formal model of Poker rules, one can more easily adapt and test the agent in new environments thus improving the overall agent's capacity and robustness in game playing, allowing it to have a much more complete strategy. The goal of this approach is to answer to even more research questions when developing Poker game playing agents.
- Interoperability between game playing agents. In fact, nowadays, most Computer Poker researchers use different technologies to develop their game playing agents, which makes it difficult to test new approaches against previously developed ones. Some simulation systems try to solve this, like [22], [77] but they only provide the API and not the communication protocol. Agents can also be assessed in the ACPC competition by AAAI [28] but its benchmark server is only available for people that participated in the last year competition. A unique rule and communication representation model would certainly allow for more proper agent assessment, while maintaining the developers' preferences regarding technology.

For these reasons we propose a new Game Description Language (GDL) for Poker games –Poker Game Description Language (PGDL) – was developed based on XML. The goal of a GDL is to describe the state of a game as a series of facts and the game mechanics as series of logical rules. GDL's are typically used by General Game-Playing

Systems (GGPS) as input. GGPS are systems that are capable of recognizing a formal description of a game and play the game effectively without human intervention, such as Zillions of Games[32].

PGDL, unlike other incomplete information GDLs, is uniquely focused on Poker agent development and testing. Therefore, PGDL was developed to only identify the key concepts of Poker games rules in order to facilitate the definition of known or non-existent Poker variants by users with Poker domain knowledge. The reason behind the creation of a Poker specific GDL is to balance the definition and implementation time of a generic Poker playing agent. The usage of a more generic GDL would hinder the development because of its lower level nature, which would make simple strategies really hard to understand and implement. With a Poker specific GDL one sacrifices the agent's capacity to play other games but agents' strategies will surely benefit of the extra domain knowledge available.

To support the creation and assessment of PGDL entities, a general game playing system was also developed. This system allows users to not only play the PGDL described game against basic agents but also provides a proof of concept API that allows for game playing agent development. The development of the PGDL simulation/game playing system was divided in the following stages:

- Identification of Poker base rules with emphasis on the differences between its variants.
- Conception of a XML based language capable of specifying the identified rule differences and the creation of PGDL instances.
- Creation of a XML-Schema that validates PGDL instances.
- Construction of a system that recognizes the XML language (in Prolog) and that is capable of generating the specified game.
- Construction of an application (PGDL Builder) that supports the creation of PGDL documents.

[32] http://www.zillions-of-games.com/

- Development of a generic game playing agent that can play competently any variant described by PGDL.

4.2.1 Poker Variants

Poker is a group of similar games with the same base rule set. The denomination for a specific set of rules is called *variant*. The variants of Poker can be divided in 3 groups:

- **Draw Poker** – each player receives a set of private cards that only he/she can see and can improve the hand by card replacement. This group of games is usually played by casual players. Examples of Poker games that are part of this group are Five-Card Draw, Badugi and Kansas City Lowball;
- **Stud Poker** – each player receives a set of exposed cards (cards that belong to the player but everybody at the table can see) and a set of pocket cards that only the player can see, in multiple betting rounds. Six-Card Stud, Razz, Eight-or-better and high-low stud are variations of Stud Poker;
- **Community Card Poker** – games in which each player receives a variable number of private cards to form an incomplete hand, which is completed by combining private cards with public shared cards (exposed to every player). The most popular poker variant nowadays, Texas Hold'em, belongs to this group as well as Omaha Hold'em and Manila.

Poker variants rules differ on the following features:

- **Number of betting rounds** – for instance, Texas Hold'em has 4 betting rounds and Five-card draw has 3 betting rounds.
- **Number of private and public cards** (and the way they are dealt) – in Texas Hold'em 5 public cards are dealt and each player receives 2 private cards, while in Cincinnati 4 community cards are dealt, one before each round of betting, and each player has 4 private cards.
- **Forced antes** – some variants force all players to bet a certain quantity of money before the cards are dealt. This amount is called ante.

- **The betting order** – there are variants such as Seven-card stud in which the first player to act is the one with the lowest exposed card and variants such as Omaha Hold'em where the first player is the one to the left of the big blind.
- The maximum **number of players**.
- **Scoring** – there are high-games in which the highest hand wins and low-games where the lowest hand wins. There are also high-low split games, where the best and the worst hands split the pot.
- **Deck** composition – there are variants that are played with only a few cards from the deck, such as Manilla (only cards above 7 with a total of 32 cards).
- Existence of **wild cards** – special cards that can score as any card (usually Jokers).
- **Replacing cards** – some variants, like Anaconda, allow players to pass cards between them in various ways. In other variants, like Badugi, players have the opportunity to improve their hand by discarding some cards and obtaining replacements from the dealer. There are also variants that force players to discard a fixed number of cards, without replacement.
- **Betting structure** – Another major difference between the variants of poker is the betting structure. The structure can be limited, pot-limited and no-limit. The limit games are the ones in which there is a fixed value for each bet made by a player. In a pot-limited game no player can raise more than the size of the total pot. In these last two structures, until winning the game there can be a limited number of raises during a round. In no-limit games there are no limits on bets.

Table 18 summarizes the main differences of the most popular and played Poker variants.

Table 18 – Differences between Poker Variants

Variant	#Rounds	Cards					#Players
		Number	Shared	Exposed	Closed	Wild	
Texas Hold'em	4	52	Yes (5)	No	2	No	2 to 9
Omaha Hold'em	4	52	Yes (5)	No	4	No	2 to 10
Baseball	4	52	No	Yes (4)	3	3/9	2 to 8
Cincinnati	5	52	Yes (5)	No	5	No	2 to 9
Five-card draw	2	52	No	No	5	No	2 to 6
Anaconda	4	52	No	No	7	No	2 to 7
Manilla	5	32	Yes (5)	No	2	No	2 to 9
Seven-card stud	6	52	No	Yes (4)	3	No	2 to 8
Kuhn	1	4	No	No	1	No	2 to 3
Leduc	1	8	Yes (1)	No	1	No	2 to 3

4.2.2 PGDL Language Specification

In this section the structure of PGDL files is described. The PDGL format is based on XML. The format is enclosed in a hierarchical description of game rounds. The description of each game round compromises the flow of the game. There are also other elements to describe generic rules of the variant (such as the number of players) or meta-information (such as the name of the variant). Figure 33 summarizes the key components of the language by presenting the tree structure of a PGDL file.

Examples of PGDL documents representing popular Poker games can be found in PGDL Documents.

4.2.2.1 Initial setup

The *PokerGame* is the root component of PGDL which includes the name, the winning determination (High, Low or Mixed), the ante value if the game is played with or without wild cards.

```
<PokerGame
   name="Leduc"
   wildCards="No"
   winningType="High"
   ante="1" />
```

Language: XML

Every *PokerGame* node must have a *Players* child node where the maximum and minimum number of players is defined.

```
<Players
   minimum="2"
   maximum="4" />
```

Language: XML

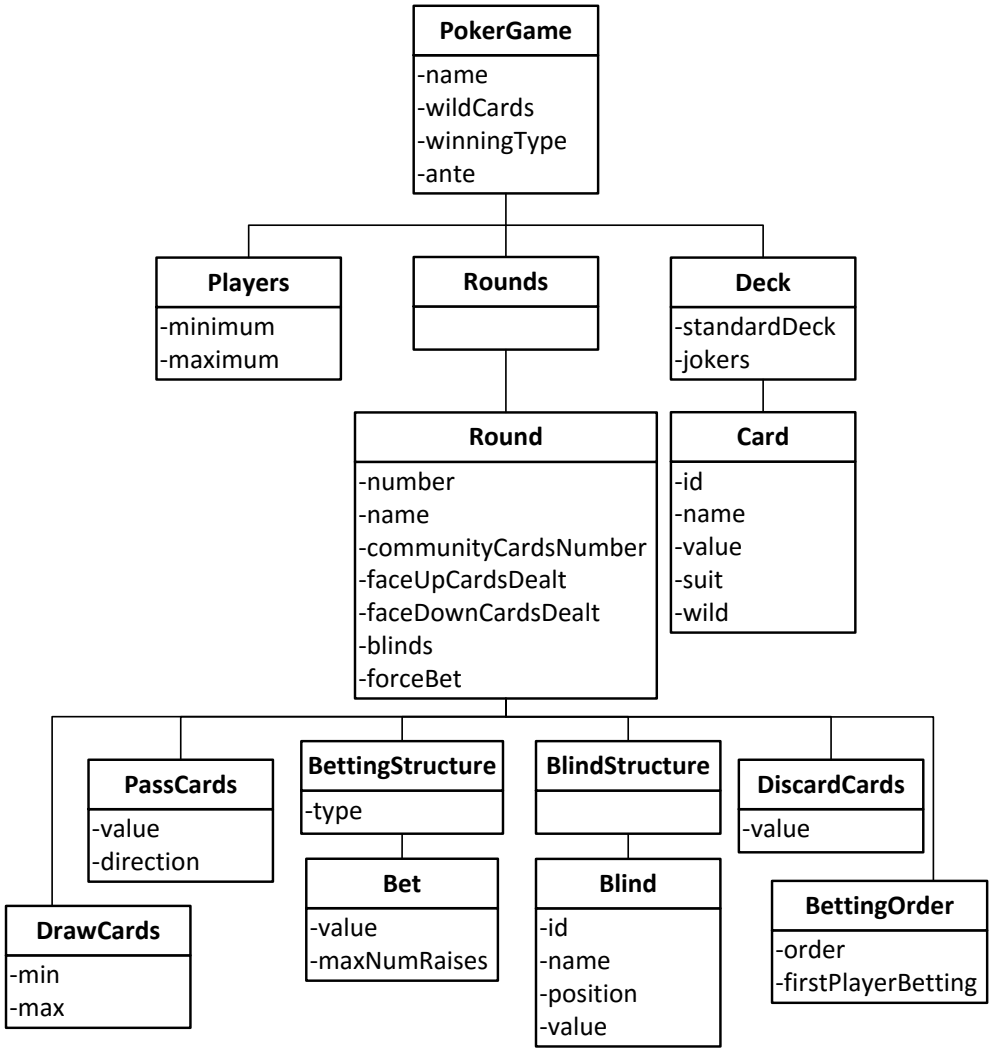


Figure 33 – PGDL Specification

4.2.2.2 Deck

Poker games can be played with a standard deck (52 cards without Jokers) or with a partial deck with a given number of Jokers.

```
<Deck standardDeck=”Yes” jokers=”0” />
```

Language: XML

If the game is played with wild cards, any card can be used as wild (usually Jokers are used as the default wild card). The deck definition allows not only using directly a standard deck but also personalize which cards belong to the deck, with custom names. This way one can even define Poker variants with two decks for instance. For each card one has to indicate the id and name of the card, the suit, its value (any value of a standard card) and if that card is wild. This representation does not cover variants with dynamic wild cards.

The example of deck for Kuhn Poker (one of the simplest versions of Poker, used mainly for research purposes).

```
<Deck standard=”No” jokers=”0”>
   <Card id=”k” name=”King” value=”K” suit=”h”
wild=”No” />
   <Card id=”q” name=”Queen” value=”Q” suit=”h”
wild=”No” />
   <Card id=”j” name=”Jack” value=”J” suit=”h”
wild=”No” />
</Deck>
```

Language: XML

4.2.2.3 Rounds

The *Round* element is the most important component of the PGDL file structure because it is associated with the game flow. It describes how the rounds will take place during the game. Each round has a round number (to control the order of rounds), a name, the number of dealt shared cards, the number of faced up and faced down cards that each player is dealt, one Boolean to control if the round must start with a bet and another one to check if the round has blinds.

```
<Round
   number="1"
   name="Round One"
   communityCardsNumber="1"
   faceUpCardsDealt="0"
   faceDownCardsDealt="1"
   blinds="yes"
   forceBet="no">
        …
</Round>
```

Language: XML

Furthermore, each round has sub-components: the Betting and Blind Structure of that round, the Cards Rules and the orders of the bets. Each round must have an individual betting structure defined.

The Betting Structure must be one of the three available types: Limit, No Limit and Pot Limit. Depending on the picked type, one has to indicate the maximum number of raises allowed per player and the bets' default value.

```
<BettingStructure type="noLimit">
    <Bet value="1" maxNumRaises="3" />
</BettingStructure>
```

Language: XML

Blind Structure only exists if the attribute *blinds* is activated (equals to 'yes'). This element contains a non-empty set of *Blind* elements. A *Blind* is described by a name, a unique id, the value of the blind and the position of the player that will post the blind.

```
<BlindStructure type="noLimit">
    <Blind id="smallBlind" value="1"
name="Small Blind" position="nextDealer" />
</BlindStructure>
```

Language: XML

Card Rules are specified by three different elements: *Draw Cards*, *Discard Cards* and *Pass Cards*. *Draw Cards* indicates the minimum and maximum number of cards that each player can draw in a round. *Discard Cards* specifies the number of cards that

each player must discard in that round. *Pass Cards* defines the number of cards that each player must pass and in which direction (clockwise or counter clockwise).

```
<DrawCards min="0" max="0" />
<PassCards value="1" direction="clockwise" />
<DiscardCards value="1" />
```

Language: XML

Betting Order it's a sub-component of the *Round*. To specify it, it is necessary to indicate in what order that round will occur (Clockwise or Counter clockwise). The first player to act is also defined in this element.

```
<BettingOrder order="clockwise"
    firstPlayerBetting="nextDealer" />
```

Language: XML

4.2.2.4 Scoring

With PGDL it is also possible to customize the Poker scoring system. To customize the score we need to add the element *Scoring* as child of *PokerGame* root node. After adding it, two options are available:

- Use the standard scoring (explained in Sections 2.5.2.2 and 2.5.4.1): there we just need to select the size of the hands used throughout the game (the standard value is 5) and put the element *standard* attribute as being true. If the *Scoring* element is not present in the document, the standard scoring will be used.
- Use nonstandard scoring: the attribute *standard* must be false and the *handSize* must still be specified. In this case we need to have several *Score* child elements with the formulas (as text under the *Subrank* child element) to assign that particular score type. The formulas have access to the card values by using $\$c_i$ where *i* is the index of the card (between 1 and *handSize*).

```
<Scoring standard="false" handSize="5">
 <Score name="high card" rank="0"
default="true" sort="true" >
  <Subrank>
   $c5.rank * 28561 + $c4.rank * 2197 +
$c3.rank * 169 + $c2.rank * 13 + $c1.rank
  </Subrank>
 </Score>
 <Score name="pair" rank="1" default="false"
sort="true">
  <Conditions>
  </Conditions>
  <Subrank>
   $c5.rank == $c4.rank?
    $c5.rank * 100000 + $c3.rank * 169 +
$c2.rank * 13 + $c1.rank:
   $c4.rank == $c3.rank?
    $c4.rank * 100000 + $c5.rank * 169 +
$c2.rank * 13 + $c1.rank:
   $c3.rank == $c2.rank?
    $c3.rank * 100000 + $c5.rank * 169 +
$c4.rank * 13 + $c1.rank
    $c2.rank * 100000 + $c5.rank * 169 +
$c4.rank * 13 + $c3.rank
  </Subrank>
 </Score>
 <Score name="two pairs" rank="2"
default="false" sort="true" >
  ...
 </Score>
 ...
</Scoring>
```

Language: XML

4.2.3 PGDL System

The PGDL system is a set of sofware applications that contemplate the following features:

- Support the creation of PGDL files through an intuitive GUI;
- Generate the user-defined Poker variants from a PGDL file or through the GUI;
- Allow the user to play and create a PGDL-specified Poker variant through a simple 2D game visualizer.

Figure 34 explains the workflow of the PGDL system. With PGDL Builder the user specifies the rules of a Poker game. That specification generates a PGDL XML Document that is validated by the PGDL XML Schema, to determine if the specification format is valid. After the validation has succeeded, the PGDL XML Document is then translated to a Prolog file that contains the terms needed to configure a generic Poker implementation in Prolog. The Prolog implementation can be extended by a very simple Agent Development API that integrates the Poker simulator described in Section 4.1. Two agents that used the agent development API are natively included: a Random Agent that picks a random action and a *E*[*HS*] Agent that plays based on the Expected Hand Strength of the current hand. After that, the game can be played in a 2D Visualizer by the user against the generated agents.

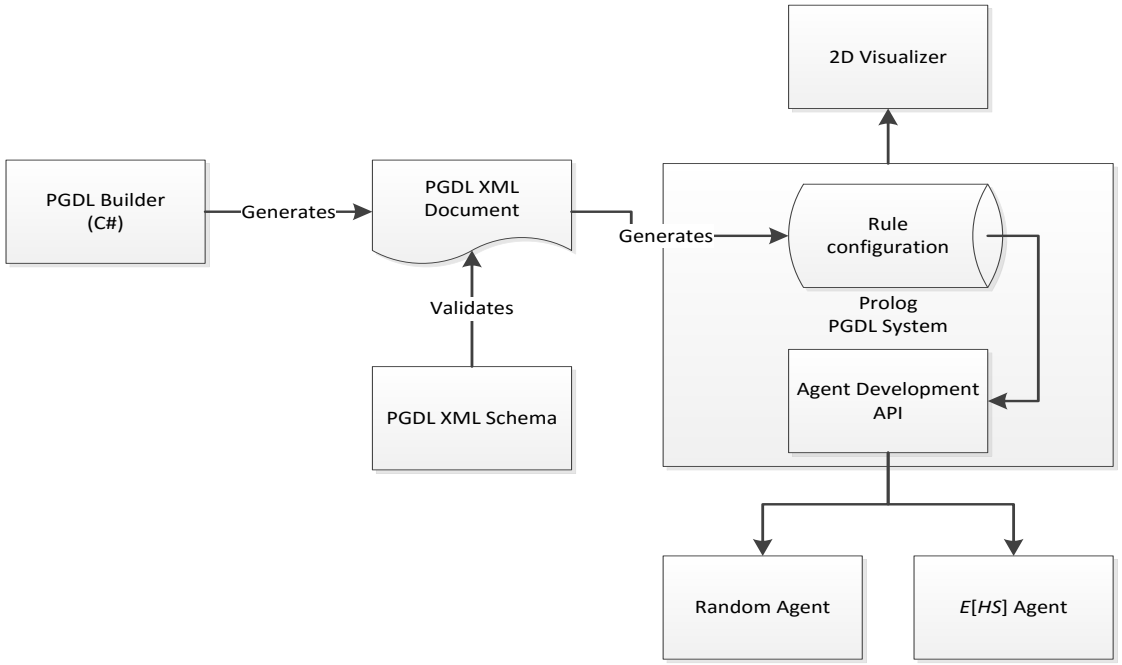


Figure 34 – PDGL Builder System workflow.

During the development of the PGDL system, several issues were addressed. In the following subsections we present implementation details of solutions to those issues.

4.2.3.1 Game rules configuration

The first problem to solve was to choose the best way to represent the list of terms in Prolog that specify the rules of a Poker variant. This set of terms was made to be accessible to support the conversion of a PGDL file to Prolog and to be easily used by

the generic Prolog system. Next an example of game rules configuration is demonstrated for the variant Leduc Hold'em (a simple variant mainly used for research purposes).

```
minPlayers(2).
maxPlayers(2).
stack(15).
name('Leduc').
winningType(high).
wildCards(0).
card(qs,'Queen of Spades',queen,spades,1,0).
card(js,'Jack of Spades',jack,spades,2,0).
card(ks,'King of Spades',king,spades,3,0).
card(qh,'Queen of Hearts',queen,hearts,4,0).
card(jh,'Jack of Hearts',jack,hearts,5,0).
card(kh,'King of Hearts',king,hearts,6,0).
round(1,1,1,0,1,'Pre Flop').
bettingStructure(1,noLimit,1,3).
blind(1,'Small Blind',1,leftDealer).
blind(1,'Big Blind',2,twoleftDealer).
bettingOrder(1,clockwise,leftDealer).
passCards(1,1,clockwise).
drawCards(1,1).
```

Language: PROLOG

A *round* is a term that is composed of six atoms: number of round (order), the ante value, the number of faced up cards, the number of faced down cards, the number of shared cards and the name of the round.

BettingStructure is a term that has four atoms: the number of the round where it belongs, the type of betting structure, the value (that is only used when the structure is 'limit') and the maximum number of raises that are allowed in the corresponding round.

The term for *card* description is composed of an id, the name of the card, the value of the card, the suit, an auxiliary value and a binary value (1 or 0) that indicates if that card is wild or not.

4.2.3.2 Representing a player state

During a game, the player is expressed as follows:

```
player(Id, Cards, PlayerType,
 PlayerAvailability, LastBet, Stack).
```

Language: PROLOG

Id is a unique identifier for the player in the game. The argument *Cards* is a list that contains the player's private cards. *PlayerType* indicates if a player is human or an agent (to allow it to be controlled by the GUI or not). *PlayerAvailability* indicates if that player is allowed to bet. The player will not be allowed to bet if it is in all-in mode or has forfeited the match. *LastBet* represents the total amount of cash that the player has bet during the current round (when a new round starts this value is set to 0 and is used to check if all player bets are matched). *Stack* represents the total amount of remaining chips of that player, in order to control the value of bets that the player can make.

4.2.3.3 Representing the game state

The game state is represented by a list that contains a list of all players, the current value of the pot which is awarded to the winning player at the end of the game, the number of raises made so far (to be used in games that limit the number of raises), a list of shared cards and the position of the dealer. The latter is used to locate the players in the table (relative positions to the dealer are used).

```
GameState = [NumberRaises-Pot-Dealer-
          SharedCards,PlayersList]
```

Language: PROLOG

4.2.3.4 Determining the end of a round

To determine if a round ended, the bet values of all available players are asserted to be the same as follows:

```
pass_aux(BetsList):-
 max_member(Max, BetsList),
 min_member(Min, BetsList),
 Max =:= Min.
```

Language: PROLOG

When this happens, the round ends and the system moves to the next round. As described in the code, the condition for this to happen is the minimum bet being equal to the maximum bet on the *BetsList*. If there are no more rounds left, the winner of the game is determined.

4.2.3.5 Determining the winner

Another problem faced was the way the winner is determined. To do this, the player with the best hand must by chosen. There are already lots of applications to compare Poker hands efficiently (described in [78]) but those are targeted to the most popular variants in which the hands are composed of at least 5 cards and a maximum of 7 cards. The fastest known evaluator is *TwoPlusTwo* Evaluator, which can evaluate about 15 millions of hands per second (see Chapter 5).

The evaluator takes a poker hand and maps it to a unique integer rank such that any hand of equal rank is a tie, and any hand of higher rank wins. *TwoPlusTwo* was used to calculate the winner in games where the hands are composed at least by 5 cards (for hands with more than 7 cards, we used the *TwoPlusTwo* 5 card lookup table and computed all combinations $C(n,5)$ of 5 cards to pick the best possible score). To compute the score of hands that are composed by a maximum of 4 cards, a new evaluator was developed (since Straights and Flushes are not possible with less than 5). To do this, we assigned a value to each possible hand based on the cards that compose that hand. For example, if we have a hand of 4 cards (C_1, C_2, C_3, C_4) and the cards are all different the way the value of the hand is calculated is:

```
numEqualValue([C1,C2,C3,C4],HandValue):-
 max_member(R1,[C1,C2,C3,C4]),
 min_member(R4,[C1,C2,C3,C4]),
 delete([C1,C2,C3,C4],R1,L),
 delete(L,R4,L2),
 max_member(R2,L2),
 min_member(R3,L2),
 HandValue is
  Rank(R1) * 1000000 + Rank(R2) * 10000 +
  Rank(R3) * 100 + Rank(R4).
```

Language: PROLOG

In this example we obtain the order of the cards 4 cards (C1, C2, C3, C4) by transforming them into R1, R2, R3, R4, where R1 is the card with the highest rank, the R2 the following card and so on. Then we just apply a different factor to transform the hand into a score, by multiplying the highest card by the highest factor. The card ranks go from 1 to 13. We selected the factors in a way to use two digits of the final results.

4.2.3.6 Dealing with wild cards

Another issue found was how to deal with wild cards when a player has in his hand wild cards and it is necessary to calculate the hand value. In that case the wild cards are identified and removed from the hand, creating a new hand. Then, the cards of the new hand are removed from the deck and with the new deck are generated all the possible combinations of the number of wild cards presented in the hand. Each one of those combinations is added to the hand and the value for that hand is calculated. The hand value is chosen from all the combinations of hands, according to the winning type of the game.

```
retrieveWildHandValue(Hand,WildCards,Value):-
 minus(Hand,WildCards,HandWWC),
 findall(C,card(C),TempDeck),
 minus(TempDeck,Hand,Deck),
 findall(Combination,
  foreach(
   in(Card, Deck),
   append(HandWWC, Card, Combination)
  ),
  AllCombs
  ),
 getValue(NewHand,AllCombs,0,Value,_Card).
```

Language: PROLOG

This prolog term represents what was exposed above. It starts by generating a hand *HandWWC* and a deck *Deck* without the wildcards with the minus operation (minus term is true if the third argument contains all elements that are on the first argument but non on the second). Next it generates all combinations, using *findall* and *foreach* terms, generating *AllCombs*. Finally it returns the maximum value of all combinations using the helper term *getValue*.

4.2.3.7 Integration with the agent development API

The LIACC's simulator agent development API is integrated with the PGDL system. To do this integration, two agents were developed: one that expands the original and abstract *PokerAgent* class from the simulator called *PGDLAgent* (see Figure 25) and that communicates through sockets with an agent developed in Prolog. The new agent developed in Prolog supports new methods (that were bridged to the agent in JAVA) that deal with information set abstraction features. The reason behind this is the fact that most Poker games usually have a very large decision tree which makes it essential to abstract information sets (by making different cases undistinguishable) to enable agents to make decisions in reasonable time. There are three types of abstraction: moves sequence abstraction, information abstraction (card set abstraction in the case of Poker) and action abstraction (more useful for No Limit games with multiple possible raise amounts to choose from).

To overcome this, in the Prolog agent implementation the following Prolog terms were added:

- *abstract_hand(+Hand,-AbstractedHand)* – abstracts the hand of the player (private and shared cards). The default term is no abstraction (*abstract_hand(H,H)*).
- *abstract_history(+History,-AbstractedHistory)* – abstracts the sequence of game actions. Again, the default term is no abstraction.
- *play(+AbstractedHand,+AbstractedHistory,-AbstractedAction)* – the actual term that is used to play. It returns an abstracted action.
- *translate(+AbstractedAction, -Action)* – translates an abstracted action to an actual action to be executed by the agent.

The generic implementation in prolog of a strategy of an agent is then defined as follows:

```
strategy(PID,SharedCards,History,Action):-
 player(PID, PCards, _,_,_,_),
 concat(PCards, SharedCards, Hand),
 abstract_hand(Hand,AbstractedHand),
```

```
abstract_history(History,AbstractedHistory),
play(AbstractedHand,
     AbstractedHistory,
     AbstractedAction),
translate(AbstractedAction,Action).
```

Language: PROLOG

4.2.3.8 Built-in agents

Two pre-built agents are included in the PGDL system: a random agent and a *E*[*HS*] (expected hand strength) based agent. The random agent picks a random action for any information set, avoiding folding (forfeit) when a check action (free pass) is possible. The *E*[*HS*] agent is based on adapted *E*[*HS*] equation (just like the Poki and Loki agents, see section 3.3). The Expected Hand Strength is the probability of the current hand of a given player being the best if the game reaches a showdown with all remaining players. For a player *i* against a giver number of opponents *n*, the *E*[*HS*] is given by:

$$E[HS]_n(i) = \left(\frac{Ahead(i) + \frac{Tied(i)}{2}}{Ahead(i) + Tied(i) + Behind(i)} \right)^n$$

EQ15

The implemented agent uses the *E*[*HS*] value to choose the action according to Table 19. For each betting structure, the agent has a fixed probability of following each action. This agent just served for testing purposes and these values were adjusted by the author's own experience of the game. They were adjusted several times empirically after enough manual observations were made.

Table 19 – PGDL in-built agent's strategy

E[HS] Value	Betting Structure								
	Limit			No-Limit					
	Fold	Call	Raise	Fold	Call	Raise 10%	Raise 20%	Raise 50%	All-In
< 30%	100%	0%	0%	100%	0%	0%	0%	0%	0%
30-50%	50%	30%	20%	50%	30%	10%	3%	2%	0%
50-80%	5%	50%	45%	5%	50%	25%	10%	5%	5%
80-100%	1%	19%	80%	1%	19%	20%	15%	15%	30%

4.2.3.9 Graphical User Interface

In order to make it easier and more intuitive for a user to specify the rules of a poker game, a GUI was developed using Microsoft C# 4.0 Windows Forms. The interface was divided in three parts: Game, Rounds and Deck. Three screenshots of each part are respectively presented in Figure 35, Figure 36 and Figure 37.

The first screenshot presents the interface used to specify the Game's general rules. In this window the user has to indicate the minimum and maximum number of players that can play the game, the way the winner is determined, the name of the game and if the game has dealer or not.

In the second screenshot the interface used to define the rounds is shown. The user has the possibility to choose the name of the round, the betting structure, the betting order, the rules that involve cards, and the blind structure where he or she can add the blinds that will occur in the game and the cards dealt. Each round is defined in different tabs. In each tab it is possible to edit that round. The order of the rounds is defined by the order of the tabs in the interface. The rounds can be re-ordered through drag & drop.

To specify the composition of the deck (third screenshot), the user has the possibility of choosing to use the standard deck in a checkbox. If not, the user has to select each card one by one from the list on the right. The user must also indicate if the game has wild cards or not. If it has, he or she has to indicate how many jokers will be used or indicate if a particular card is wild or not.

To create the game the user has to click in the "Create Game" button. If the specification has errors the user will be notified. If not, the XML and Prolog file with the specification of the rules of the game will be created and the button to play the game in the simulator 2D visualizer (see Figure 31) will be available.

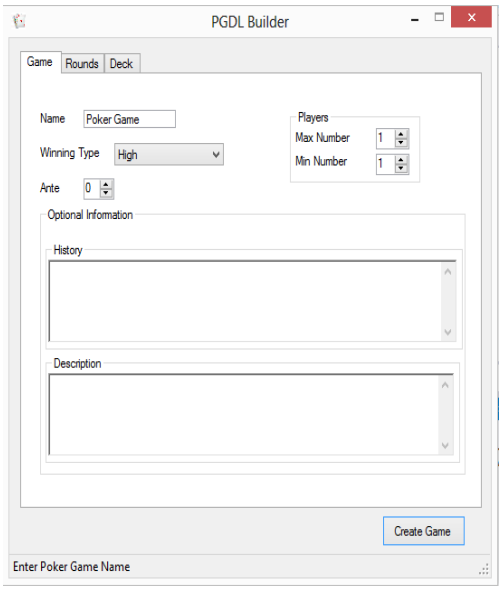


Figure 35 – PGDL GUI Games Module

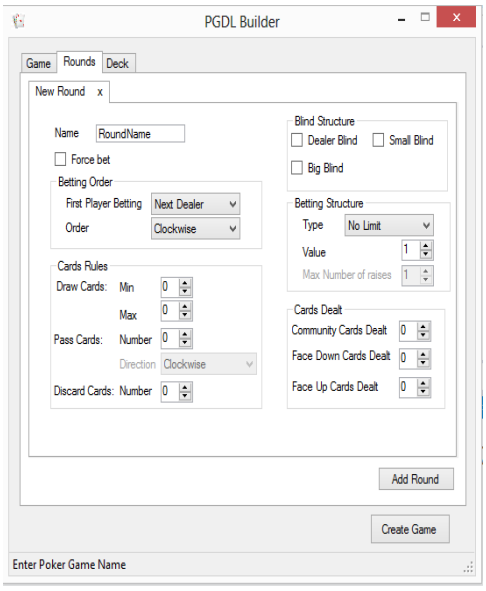


Figure 36 – PGDL GUI Rounds Module

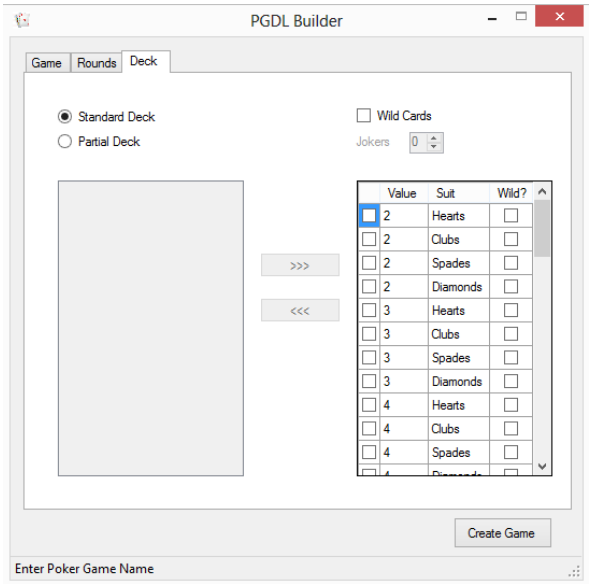


Figure 37 – PGDL GUI Deck Module

4.2.4 System validation

To validate the PGDL system, several tests were performed. First several popular Poker variants were implemented to confirm that the PGDL specification was sufficient to describe them all (see examples in Appendix C). Next, we tested the *E*[*HS*] agent against the random agent to assess if it is competent enough against the most basic agent – the random agent. Finally, we tested the GUI with several users to assess if the system is user-friendly to implement Poker variants.

The following Poker variants were implemented successfully with the PGDL specification: No-limit / Limit Texas Hold'em, Kuhn, Leduc, Cincinnati, Five-card draw, Anaconda, Manilla and Seven-card stud.

To check if the GUI is user-friendly and intuitive, usability tests were performed (Table 20). The test consisted of users (16 subjects in our tests, with at least some previous knowledge about Poker) implementing two simple variants of poker: Kuhn Poker (2 times, one with standard deck and one with 3 card deck) and Leduc Hold'em Poker. All subjects were able to complete the task with an average time of 3:42 minutes (with our without help). By analysing the results, the time spent by the users doing the tests was very similar (standard deviation of 76 seconds).

Table 20 – PGDL usability tests

Needed time (secs)		**Main issues**	
Kuhn	**Leduc**	**Kuhn**	**Leduc**
150	120	Miss game name	Miss game name
420	270	-	Community Cards vs Face Down Cards
150	150	-	-
145	180	Betting structure	Add rounds
210	240	Bet values	Community Cards vs Face Down Cards
200	150	Betting structure	-
160	210	Max raises	Community Cards vs Face Down Cards
174	240	-	Deck
350	412	Deck	Deck
253	300	Nomenclature	Betting Structure
184	266	Number of cards	-
243	230	Missed blinds	Rounds
240	296	-	Rounds
122	116	-	Rounds
230	245	Miss game name	-
Avg: ~3:35 min	**Avg**: ~3:48 min	**Total Avg**: 3:42 min	

The biggest problems faced in the GUI usage were related to the understanding of the Poker specific nomenclature, even for users that said that they played Poker regularly. This is due to the fact of most Poker variants being unknown even for regular

Poker players (the most played are Limit and No Limit versions of Texas Hold'em and Omaha Hold'em).

4.2.5 Summary

PGDL is a generic system for creating poker variants. A XML dialect was defined to represent the specification of most known Poker variant rules. From that specification, the developed system can generate a playable implementation of the game in Prolog. All of the most popular Poker variants are implementable within our system, proving its usefulness. The results of tests showed that the interface is user-friendly, well designed and is easy to use, as shown by the similar time to specify the same poker variants. This approach can enhance not only the easy implementation of any poker variant but also the creation of new variants. For future work, the system could benefit from a general implementation of the Counterfactual Regret Minimization algorithm in order to generate Nash Equilibrium strategies for any specified variant thus providing very competitive agents with the system. The biggest difficulty of that implementation would be the creation of a generic abstraction system for any Poker variant (see Chapter 5 for some pointers on this).

4.3 Poker Bot

4.3.1 Goals

A Poker Bot is a software application to serve as an interface between a Poker Software Agent and a Poker Online Casino. This kind of software enables developers to have their agents playing in real time online environments, without their adversaries knowing that an agent is playing – this is especially interesting because previous assessments of having Poker agents playing against humans, were with the human players' knowledge that they were playing against a bot. This is very important because this way the human players will be playing with their regular strategies without modifying or adapting them to play against a bot.

The development of the LIACC Bot was divided in several steps. Due to difficulty of generalization and development of this kind of applications, the recognition only works on No-Limit Texas Hold'em Games. To support its development, OpenCV was

used as well alongside a library to wrap it for C#. As explained in Section 3.9, currently Online Casinos do not support officially the use of bots, and they do not provide APIs to do it. Moreover, most of them also actually try to stop bots from playing in their software. For this reason, to build a bot several steps are needed to overcome this reality:

- Since there is no API, the bot must interact with the regular user interface. Therefore it needs to apply image processing to the interface window in order to extract the information about the current state of the game.
- The bot must control the mouse and the keyboard to be able to click on the interface controls.
- Some Casinos actually record the interaction between the user and the client by taking screenshots or recording small videos. For this reason, the mouse movement must be similar to the way humans use it.
- Casino software usually scans for the user's pc to try to find suspect applications that maybe bots. Therefore, the bot must run in stealth mode in order to avoid detection i.e. it must disguise itself as being another application (in this case a calculator).

The development of the bot was divided in the following steps:

- Card recognition
- Chips and bet amounts detection
- Human like interaction with realistic mouse movements
- Avoid detection: pressing randomly the interface buttons in random positions, random waiting time between plays...
- Deal with Casino client software updates that change the position of interface buttons, colours…
- Integrate the bot with the agent API described in Section 4.1

4.3.2 Card Recognition

The first problem to solve to address in card recognition is to identify the regions of the application interface where the cards of the player and the community cards were placed. The first approach was to use an *edge detection* algorithm, using the cards white background and their contours. However, due to random card occlusions on the application interface (sometimes due to animations) and different displaying style, this method did not have good results – Figure 38 shows an example of this problem where one of the community cards had the chips overlapping it which made the *edge detection* algorithm to fail. The algorithm that detects the card regions is illustrated on Figure 39.

Figure 38 – Card position recognition – the chips occluded the third card

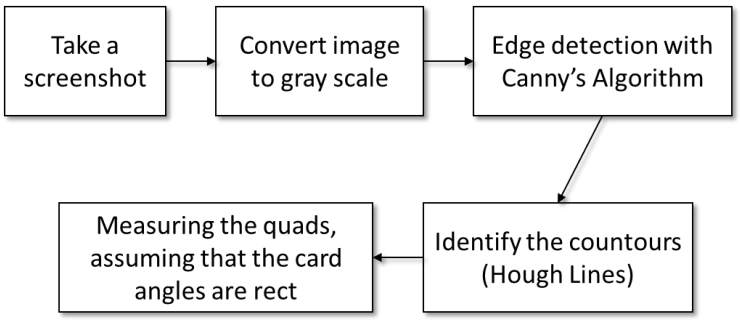


Figure 39 – Detecting cards regions algorithm

Due to the method's poor success rate (it failed about 25% of the times), it was decided to just use configurable regions with fixed positions, that is, the user of the bot has to configure it to select the desirable positions (select the square regions of the positions of the 20 player's cards and the positions of the 2 player's cards). This last method is error free (100% accuracy) but has the disadvantage of requiring the user to update the card's positions when the casino software is updated.

After getting the cards position, the following step is to guess which card it is. For this, the approach was *template matching,* i.e. having a classifier trained with all card templates in order to match the new ones that appear. Only the top left part of the card region was considered, as demonstrated in Figure 40. The reason behind this is that the selected region contains enough information for the card recognition (rank and suit). Using the whole card would not only take much more time (since the image has more pixels) but would also cause more errors. The detection of the top-left region is also by its relative width (27.35%), relative height (43.26%), relative starting position (6.89%, 8.28%) so only that part of the image had to be selected.

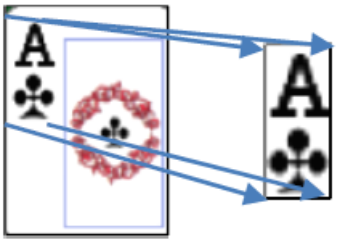


Figure 40 – Cutting the card for recognition.

After cutting the interest region we only have to compare it with all stored templates as demonstrated in Figure 41. One important thing to add is that the templates are grouped by colour density (red and black). This helps in the suit detection because, for instance, the spades and the hearts symbol is somewhat similar. If no template is matched, another screenshot is taken and the process starts all over. If that fails again, two options can occur:

- a random card is considered
- the card that leads to the worst score possible is considered

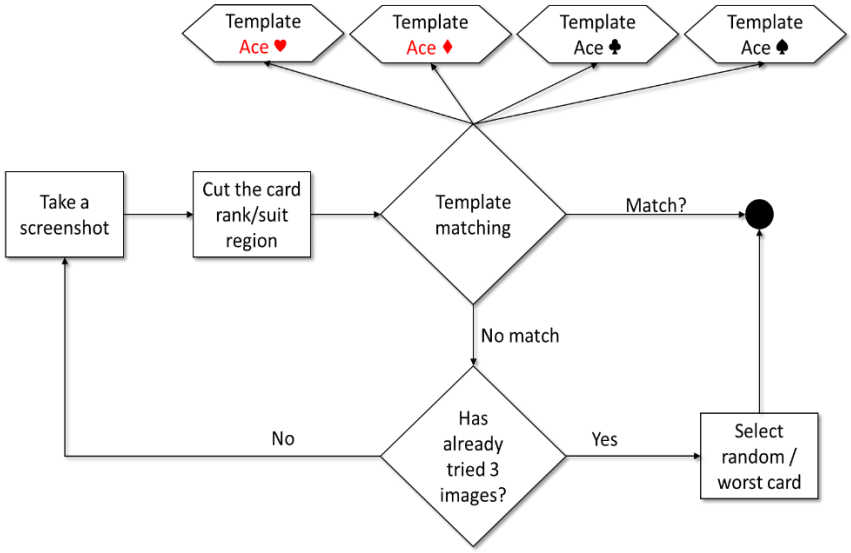


Figure 41 – Detecting the card template

4.3.2.1 Tests

To test this approach 200 screenshots were taken with two different resolutions and the algorithm was run on every screenshot. The detection rates presented in Table 21 were very good. The suit recognition is not presented on the table because when the card rank was correctly identified, the suit was also. All algorithm responses were manually verified and double-checked.

The detection rates on the lower resolution were 100% correct. However, the detection on the higher resolution failed sometimes. One possible reason behind this is that the templates were made from screenshots at the lower resolution which means that the OpenCV *template matching* algorithm has to resize the sliced images provided. Since that the sliced cards resolution is very low (about 12 x 32 pixels), that could be the reason why this happens. Nevertheless the creation of new templates solved the problem.

Table 21 – Card detection rates

Card	Image Resolution	
	1016×728	1158×826
Ace	100%	93,9%
2	100%	86,7%
3	100%	97,5%
4	100%	97,7%
5	100%	86,2%
6	100%	95,0%
7	100%	100,0%
8	100%	92,6%
9	100%	97,6%
Ten	100%	90,9%
Jack	100%	100,0%
Queen	100%	96,6%
King	100%	95,9%
Average	100%	95,1%

4.3.3 Game State Recognition (dealer button position)

Another important part of the interface recognition is the game state recognition. Without it, the agent would be playing blind.

First, the current round of the game is identified. This step is rather easy to do because the positions of the community cards are pre-established, like it was referred in the last section. By that, we just have to detect if a card is there or not, by detecting the density of the white color. If we have no cards then we are at the Pre-Flop round, if we have 3 we are at Flop, 4 at Turn and 5 at River. This information is double checked with the bot knowledge. The bot itself reads and registers every play, so it knows the current state of the game just by the actions. This method is also used to detect the end of game, by detecting a new Pre-Flop round.

Besides the round, one very important asset is to detect the position of the dealer button (see Figure 42). The approach to detect the dealer button is similar to

the one used with the cards: the user pre-selects the regions where the dealer button could appear and then every position is compared to the dealer button template (using *template matching*). The first position to identify the dealer button is the one to be assigned in the internal bot game state. This detection is important in order to correctly identify the blinds values and the order of plays.

Figure 42 – Detecting the dealer button position

4.3.3.1 Results

The same process was applied to validate the detection of the dealer button. 200 screenshots were used and the detection algorithm was run on all of them. The detection rate for the dealer button was 100%.

4.3.4 Recognize betting amounts and actions

The recognition of the betting amounts was by far the biggest challenge in developing this bot. In the used casino client the betting amounts are drawn by chips of different colours with each colour representing a different value. Detecting the amounts through those images would be very difficult because the number of occlusions is very high and the chips are very small for this to be a viable solution (see Figure 43).

Figure 43 – Chips representation in the casino interface software.

The solution that was followed was to use *OCR*[33] functionalities of OpenCV, more particularly the incorporated module called *Tesseract*. This approach was used not only

[33] OCR – Optical character recognition

for the chip amounts but also to detect the players' actions (which can be encountered below the nickname part – see Figure 44). The player's actions could be either represented by amounts or sentences in Portuguese. The same approach was followed as for the cards: positions of interest are pre-defined by the user. The followed algorithm is represented by the diagram in Figure 45. One important thing to clarify is the image scale on the 3rd step. This image scale was made to increase the accuracy in character recognition. The average detection rate for each image scale is presented in Figure 46. As it can be seen, the detection success rate seems to stabilize for a scaling of 2.3 times. This means that there is no reason to scale over that since scaling also means processing an image with a higher number of pixels.

Figure 44 – Action representation in the casino interface software.

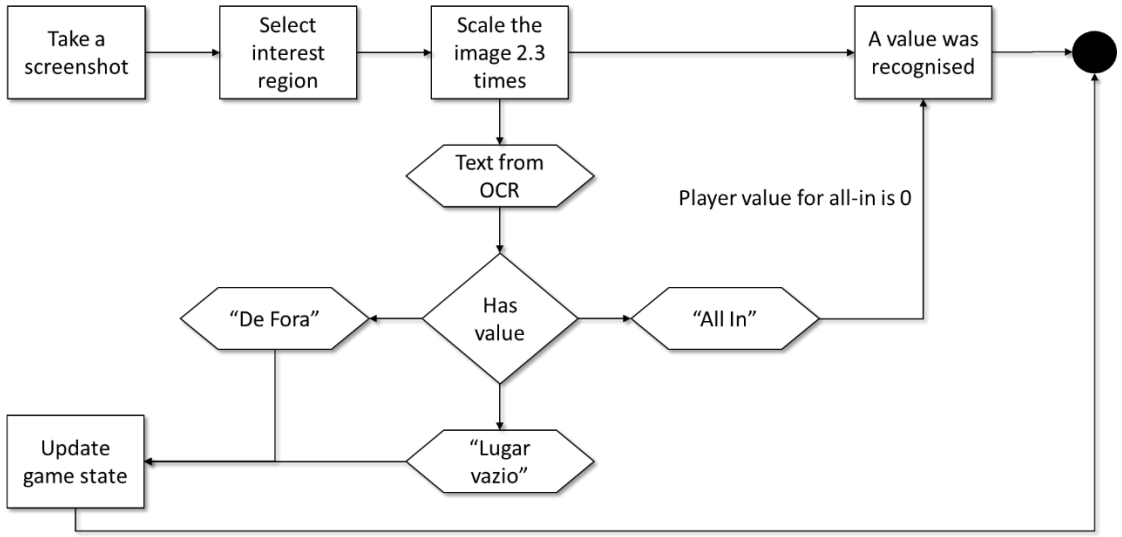


Figure 45 – Action and bet amounts detection.

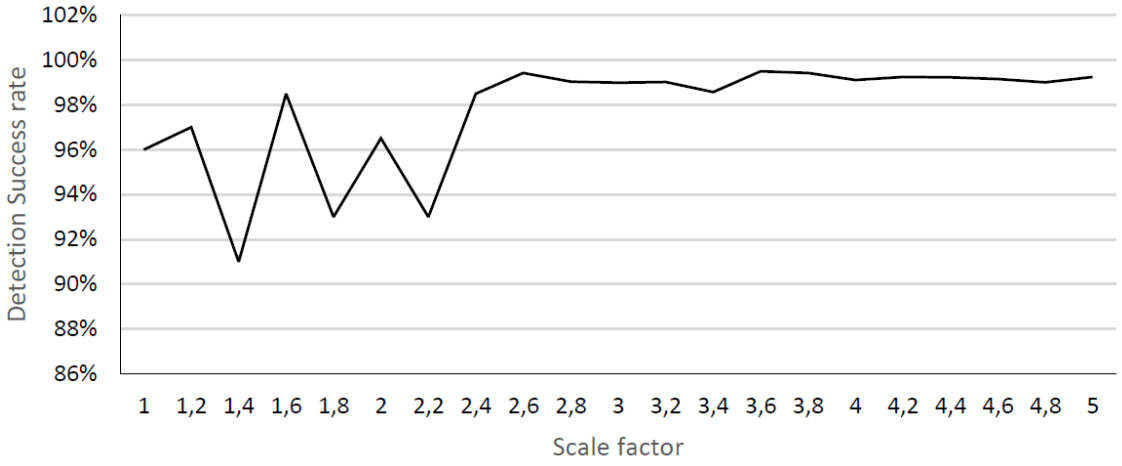


Figure 46 – Average detection rate per scale factor

4.3.4.1 Results

The process for assessing the results was also the same as it was used in the last two sections. The same 200 screenshots were used and the algorithms response was manually compared and double-checked. The detection rate was not as good as it was with the cards but it was still accurate – please observe Table 22 for details. As it happened with the cards, the detection in higher resolutions performed slightly worse. However, for the global amount of detections the average was still the same (the detection on 1016×728 had much less "Fold / without player" messages to detect. The "Fold / without player" message was also the most difficult to detect but, however, the one that has less impact on the game. This is so because it is possible to easily detect a player that is not placed on that position or that is folding because the chips near the player disappear when this happens.

Table 22 – Amount detection rates

Type of amount	Image Resolution	
	1016×728	1158×826
Bet amount (number)	100%	99,5%
All-in	100%	93,5%
Fold / Without player	87,9%	78,6%
Average Global	98,7%	95,1%

4.3.5 Simulating human behaviour

In order to simulate human behaviour on the interface, two things were done:

- Sending messages to the chat
- Simulating realistic mouse movement

Sending messages to the chat was very straightforward. In order to not always send the same text messages, an approach similar to the *"Lero-lero generator"*[34] was used (with more appropriate sentences).

Simulating realistic mouse movement was based on the *Bezier* curves algorithm. The Bezier curves have control points that, depending on the function degree, can transform a line into a curved line, where parts of the line deviate from their original trajectory to approximate the control points (see Figure 47).

Figure 47 – Bezier curve example between points A and B (degree = 2).

The following equation can generate a *Bezier* curve, where *P* are the control points and *t* is the time resolution.

$$P(t) = \sum_{i=0}^{n} {}^{n}C_i(1-t)^{n-i} \times t^i \times P_i, with\ t \in [0,1]$$

EQ16

The approach followed to define the points where the mouse must pass is in Figure 48. One important thing that was added was some noise to the trajectory. This will make the movement less precise, just like humans do. Moreover, the speed of the movement is also controlled – at the first moments the mouse moves more quickly and at the end the mouse moves more slowly (human players usually do that, first they

[34] http://www.lerolero.com/ - a website that generates random Portuguese sentences similar to Lorem Ipsum. It combines 5 parts of sentences that all connect to themselves.

move the mouse quickly to reach a region near the target and then they just adjust it slowly. The creation of the curve follows the following parameters:

- If the distance is below 80 pixel, no control points are used (straight line)
- If the distance is between 80 and 200, 1 control point is used
- If the distance is above 200 and below 400, 2 control points are used
- If the distance is above 400 and below 700, 3 controls points are used
- IF the distance is above 700, two Bezier curves are used with 3 control points each.

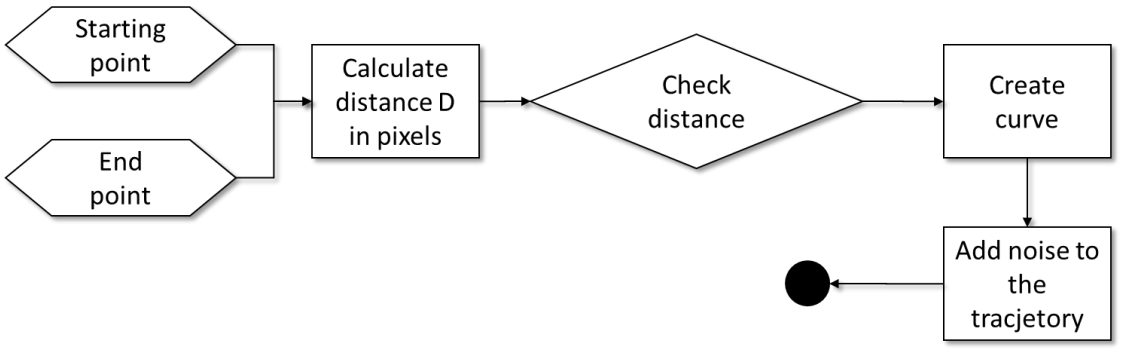


Figure 48 – Computing the mouse movement trajectory from one point the other

To validate the mouse movement methodology, several mouse movements were recorded into two movies: some from the agent and other from human users. 27 test subjects were asked if they could identify the bot mouse movement. 23 test subjects were able to identify which movie represented the agent, but 55% of them needed to watch the videos for a second time. Despite these results not being good, there are several things that conditioned the tests, but the main one was that they knew that one of the videos was from a bot. Table 23 summarizes the test results for this asset.

Table 23 – Identify mouse movement

	Only saw the video once	Saw the video twice	Total
Identified the bot	8	15	23
Couldn't identify the bot	3	1	4

4.3.6 Graphical user interface and limitations

In order to help the use of the bot, a simple graphical user interface that shows online the state of the game was implemented (Figure 49). The application has a configuration module to select (through screenshots) all the positions of the parts that needed to be identified (cards positions, possible dealer button positions, player amounts positions, etc.). The stats module is also useful – since this is an automatic player it is very important to check regularly its profit evolution.

For now, this main limitation of this agent is that it is incapable of selecting the Poker table where it is going to play (that must be selected manually by clicking on the *"Run Agent"* button on the interface). This should be improved in future version of this application, but it also requires algorithms that appropriately choose the best room that is more fitted to the agent's level of skill. Another improvement point is to make it able to run on stealth mode (so it is not detected by the software random screenshots). This could be done now by running the casino client inside a virtual machine, but adding to the software would be a great feature. However, during the several hours of tests the bot was never detected.

The resulting implementation is not perfect in its detection mechanisms and may confuse the software agent by giving it an non accurate information set. However, as the results demonstrated its accuracy is already very good for this kind of application. The implementation of this system enabled the results presented on Chapter 7 where, for the first time ever reported, an agent showed that it was possible to win money online consistently against human players without them having the knowledge that they were playing against a software agent.

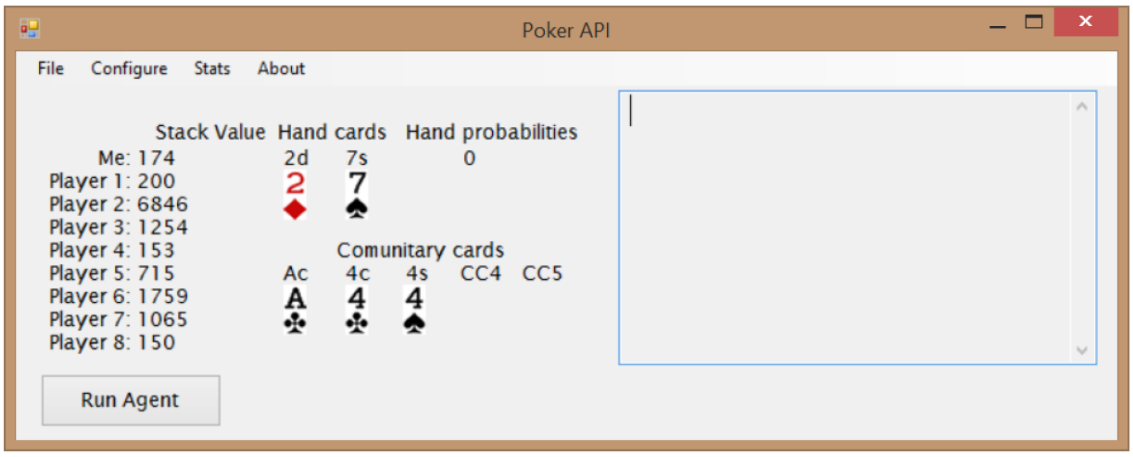


Figure 49 – Poker Bot user interface

4.4 Summary

This chapter described the tools (and their evaluation) that were developed within this thesis work. They were built to support not only this thesis's work but also enhance future developments in the Computer Poker and stochastic incomplete information games domain.

Chapter 5

Abstraction Techniques

This chapter describes the improvement of Poker related abstraction techniques that lead to the creation of two new methodologies – **Average Rank Strength (ARS)** – which is an improvement of the Effective Hand Strength formula and **Reduced Game Utility Abstraction (RGU)** – which is a more generic method that can be applied to other games (in Poker particularly, it does not require the Hand Strength).

5.1 Definition

Abstraction is the process of reducing the game search space by combining and grouping knowledge. In more practical terms it means having a group of decision points or conditions where we decide to act the same way when different, although similar game conditions are in place. One good example of this is to imagine defining a full rule based strategy for Texas Hold'em Poker (see Section 3.2.1). If no abstraction was done, it would be impossible to do so. But by saying something like *"go all-in when you have two pairs or more"*, we are already defining an action for a lot of possible hands – which is something that humans do naturally by instinct.

In terms of the game of Poker (depending on the game's rules) three types of abstraction are usually considered:

- **Card abstraction:** for software agents, the cards are usually grouped into very small groups – this is commonly known as *bucketing*. For instance,

the first approaches of Nash-Equilibrium based agents grouped all card strength combinations into 20 different buckets (different bucket sets were used for each game round), which is very different than the numerous amount of possible scores that exist (however, even with 20 buckets, the length of the search trees were still enormous). Most common abstraction approaches in the last years are based on the Hand Strength Formula (see Section 3.5.1) which has a big problem – it is a simulation method that considers that the probability of playing any hand is equal.

- **History abstraction:** this consists on abstracting betting sequences. A betting sequence is an ordered list of actions that can lead to a stage in the game. E.g. **'ccr'** means that the first player called, then the player next to it called as well and the next one raised. Betting sequences always lead to the same state, round and acting player – only the current conditions (private and community cards) may change the action that should be selected. One possible way of combining betting sequences would be to replace previous rounds plays by the pot value (like any abstraction, this introduces an error – the way a certain amount of the pot is reached might reveal details about the opponents holdings).

- **Betting amounts abstraction:** abstracting the bet amounts is only applicable to No-Limit versions of Poker, where the value of a raise action is continuous. If we try, for instance, to build all game sequences for a No-Limit game, that would be unfeasible. So, raise action values must be grouped. In Chapter 7 the amount of raises extracted from some game logs could give a hint for betting amounts abstraction, by considering clustering the relative betting amounts into same sized groups.

In this thesis we only address card abstraction, by developing two new techniques.

5.2 Improving Current Algorithms

In order to improve abstraction techniques for Poker, the first step was to try to improve already existing methodologies. Most methodologies are based on hand rank and odds comparators (see Sections 3.4 and 3.5). First the hand rank evaluators were compared in order to check which one should be used. Hand rank evaluators are very important because they transform the hand rank into a number, being therefore much easier to deal with it. One characteristic of hand rank evaluators is that the higher the number is the higher is the rank. Hand rank evaluators also distinguish between sub-ranks within a rank group (e.g. all possible pairs and their combinations with the several card kickers).

5.2.1 Hand Rank Benchmark

In order to determine the fastest hand rank evaluator, a benchmark test was performed. To provide a fair assessment, the test consisted of ranking a pre-computed sequence of all possible combinations of 5 card hands (2,598,960 hands). The tests were performed 1000 times each on an Intel I7-3940XM CPU (8 cores) with or without parallelization. The set of hands was tested with each described hand rank evaluator(s) and the results are presented in Table 24.

Table 24 – ***Hand rank function benchmark***

Hand rank program	**Average elapsed time for 1.000 trials in milliseconds**	
	Non parallel	Parallel
Cactus Kev	807.13	591.22
Paul Senzee	403.04	195.44
Pokersource	2,520.44	980.14
TwoPlusTwo	91.09	37.98

From the tests, it is possible to verify that the TwoPlusTwo Evaluator is by far the fastest hand rank evaluator, performing the same calculations in at least roughly a quarter of the time needed by the other evaluators. After testing and using each evaluator, we also identified the main advantages and disadvantages. Below follows a table (Table 25) summing up the qualitative features of each evaluator, which demonstrate as well that the TwoPlusTwo evaluator is not only the fastest evaluator

there is, but also the one with the best features (except from memory, but in today's computers 80Mb of ram is not a very serious limitation).

Table 25 – *Hand rank comparison*

	Memory	Speed	Usage	Hand's size		
				5	6	7
Cactus	< 1Mb	++	Normal	×		
Paul	266Mb	++	Normal	×		×
Pokersource	n/a	+	Hard	×	×	×
TwoPlusTwo	80Mb	+++	Easy	×	×	×

5.2.2 Hand Odds improvement and benchmark

While hand ranks are important, they are not directly used by abstraction techniques since there is not necessarily a clear inter-association between the produced integers by the hand rank functions (they are more used by simulators to assess the game's winning agent). However hand rankers are essential to produce the hand odds algorithms like Hand Strength or the most currently used: **Expected Hand Strength** *E[HS]* (see **EQ17** or **EQ15** for the simple form).

$$E[HS]_n(P_i, S) = \left\{HS_n(P_i, S + x) : x \in [\mathrm{D}]^{5-|S|} \wedge x \not\subset (P_i \cup S)\right\}$$

EQ17

The expected hand strength *(E[HS]n*) [79], also known as equity, is the probability of the current hand being the best if the game reaches a showdown, with all remaining players. It consists of enumerating all possible hands that an opponent can have and all possible unveiled shared cards. This methodology is very similar to the Hand Strength (HS), but it is far more accurate because it considers the score value that can be effectively used at the end of the game. However, the number of iterations needed by E[HS]n is much higher than for HS, making it a much slower option.

This method can be improved by using Monte Carlo. To do that, EQ17 was changed to sample possible board and opponent cards instead of enumerating them all, so instead of *x* belonging to all combinations of size 5 of the deck (with exception to the player's private cards), *x* belongs to a subset of that superset with quadratic random sampling (with higher probably for higher cards). The obtained results are on Table 26.

Table 26 – *Sampling board cards in E[HS] algorithm*

Number of Samples	Number of iterations	Error
All samples	$\approx 3.17 \times 10^{11}$	0
10000	$10^5 \times P_{45,4}$	~0.0005
1000	$10^4 \times P_{45,4}$	~0.001
100	$10^3 \times P_{45,4}$	~0.012
10	$10^2 \times P_{45,4}$	~0.151

As it can be seen, the best number of samples to use in a Monte Carlo sampled version of *E[HS]* is 1000, because it already produces a very small error for the small number of iterations. The decrease of the error rate per number of iterations follows approximately a Chi-Square distribution (with 1 degree of freedom: k = 1, also see **EQ18**). Around the 1,000 iterations point the error decrease rate is so small, that it is not worth to increase the number of iterations.

$$f(x) = \frac{x^{\frac{k}{2}-1} \times e^{-\frac{x}{2}}}{2^{\frac{k}{2}}\Gamma\left(\frac{k}{2}\right)}$$

EQ18

5.2.3 Average Rank Strength

In order to solve the efficiency problems of the previously presented methods, we introduce a new method called Average Rank Strength (*ARS*). This method consists of using the hand rank to estimate the future outcome of the match, without having to generate all card combinations. This is simply done by storing the average value of the *Odds* function in a look-up table, indexed by rank. Since there are only 7462 possible ranks, it is completely feasible to store pre-computed average future ranks in memory.

Storing the *Odds* values for each rank is not enough, since it is necessary to identify the player's private cards. To better illustrate this, let us analyse the following hand: A♣ A♦ A♥ K♥ K♠. This hand always scores a Full House regardless of which two cards belong to the player. However, depending on which two cards belong to the player, the odds can be different: if the player has the two Kings, an opponent could have the remaining Ace, thus being ahead of the player; if the player has two Aces, then victory is assured.

This problem was addressed by introducing a second dimension in the look-up table – the pocket hands id. The pocket hands id is a unique number for a pair of cards, which takes into consideration game isomorphisms. The total number of possible starting pair ids is 169 (this is the maximum number of possible combinations of two private cards with different meaning, considering the suit rotation: e.g. A♣ A♦ is the same as A♦ A♥). To quickly obtain the pair id, the ids are stored in a 52 × 52 pre-computed table named *pairs*. Thus, the id of a pair $P = \{Card_1, Card_2\}$ is given by $pairs[Card_1][Card_2]$.

Giving the described look-up table structure, its total size is 7462 × 169 × 8 bytes ≈ 9.62 MB, where 7462 is the number of possible card ranks, 169 the number of unique pairs and 8 the size of a double (in most machines).

This approach was tested against the approximate computation of the $E[HS]_n$, since it is the most common used technique. Moreover the TwoPlusTwo rank table was also needed to compute the index to search in the Average Rank Strength lookup table (since it is the fastest rank evaluator). Since TwoPlusTwo returns an index between 0 and 36874 with sparse values (only about 20% of the table values are used), an auxiliary table was created (similar to the *pairs* table) to convert the TwoPlusTwo rank to an index between 0 and 7461. This way we reduced each table's size from 50 MB to the 9.62 MB.

The *ARS* value for a given position is given by EQ19, where n is the number of opponents, I is the index of the pair of cards, R is the integer value of the rank. The *ARS* lookup process and architecture is summarized in Figure 50. Both *x* and *y* are card hand iterators in this equation: they iterate respectively over all combinations of 2 and 5 cards. x_1, and x_2 are respectively the first and the second card of the hand *x*.

$$ARS_n(I,R) = \overline{\left\{\begin{matrix} E[HS]_n(x,y): x\epsilon[D]^2 \wedge y\epsilon[D]^5 \wedge \\ x \not\subset y \wedge Rank(x,y) = R \wedge pairs[x_1][x_2] = I \end{matrix}\right\}}$$

EQ19

Since this method introduces an error, we included in our architecture the possibility of using a stochastic response. The look-up error δ_n for *ARS* is given by:

$$\delta_n(P_i, S) = \left| E[HS]_n(P_i, S) - ARS_n\left(pairs[P_{i_1}][P_{i_2}], rank(P_i, S)\right) \right|$$

EQ20

The maximum $\delta_{n,max}$ and minimum $\delta_{n,min}$ look-up errors for each *ARS* position can then be stored in two additional look-up tables of the same size. This way, when consulting the *ARS* table we sum-up a random variable in the interval [$\delta_{n,min}$, $\delta_{n,max}$] to the value stored in the table.

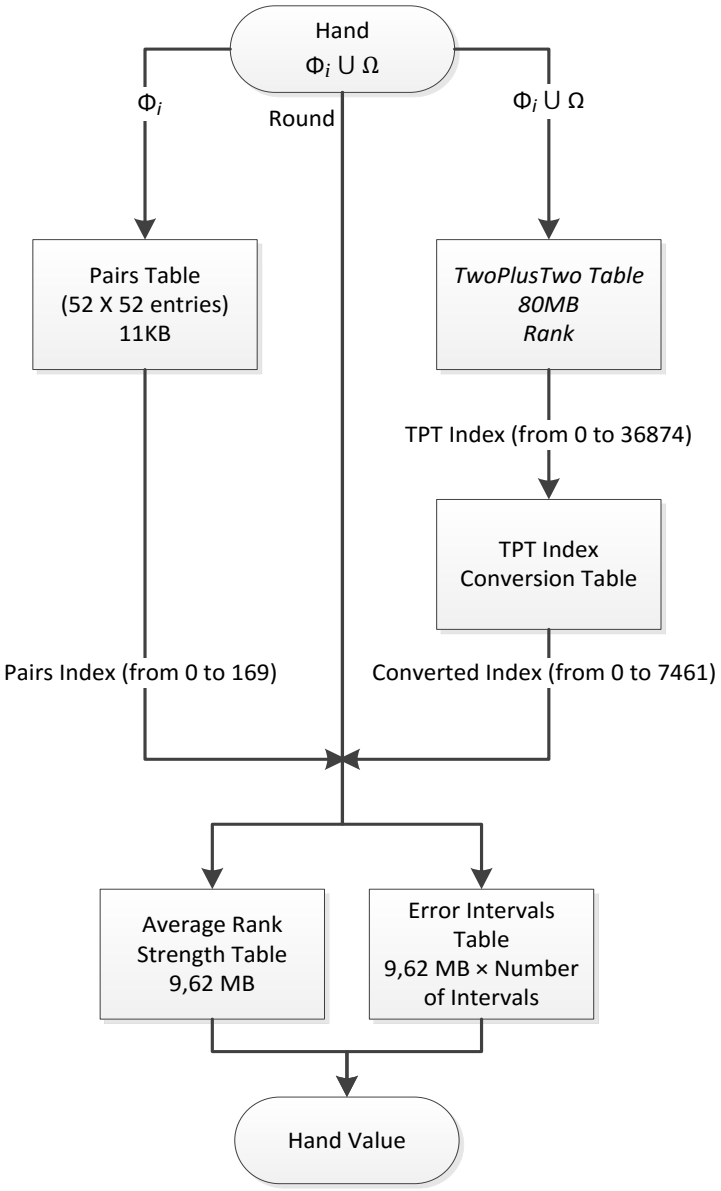


Figure 50 – ARS tables lookup process and architecture.

It is also possible to increase the precision of this methodology by having more than 2 additional look-up tables. These look-up tables can be used to store the discretized intervals of the error's distribution by setting them with percentile medians.

5.2.3.1 Results

In order to determine the speed-up factor of the new method against the *E*[*HS*] method, a benchmark test was performed. The test consisted of ranking a pre-computed sequence of 1,000,000 hands with 7 cards each. The tests were performed 1000 times each on an Intel I7-3940XM CPU (8 cores) and are presented on Table 27. The obtained standard deviation from the mean of the presented values is negligible in all cases. The results described in Table 27 did not take into account the δ_n correction tables.

Table 27 – *Benchmarking Average Rank Strength*

Hand rank program	**Round**	**Average elapsed time for 1000 trials in seconds**	
		Non parallel	Parallel (8 cores)
Expected Hand Strength (E[HS])	Flop	387.71	108.90
	Turn	309.18	90.19
	River	263.79	75.98
Average Rank Strength (ARS)	Flop	0.32	0.06
	Turn	0.41	0.09
	River	0.43	0.10
Speedup factor	Flop	1211.59	1815.00
	Turn	754.10	1002.11
	River	613.47	759.8

Our benchmark test demonstrates very promising results, with an average speed-up of 1,044.24 (about 1,000 times faster). Poker agent strategies that use abstraction based *E*[*HS*] can benefit from this speed improvement. Methods such as CFR [23] need to perform these calculations billions of times.

5.2.3.2 Error analysis

We also analyzed the difference between this method and the hand strength method. The heat maps for *E*[*HS*] and *ARS* at the River round and against 1 opponent are in Figure 51.

This approach not only provides a much faster response to queries – about three orders of magnitude faster – but also it does so with negligible error, as can be seen from the heat maps in Figure 51. At the River, the average absolute difference between the two methods is 0.011, the maximum difference found was 0.062 and the summed squared error is 0.039. The use of a stochastic response with the δ_n correction tables also improved these results. The average maximum error found was 0.02 and the average is less than 0.001, with a speed-up reduction of only about 8%.

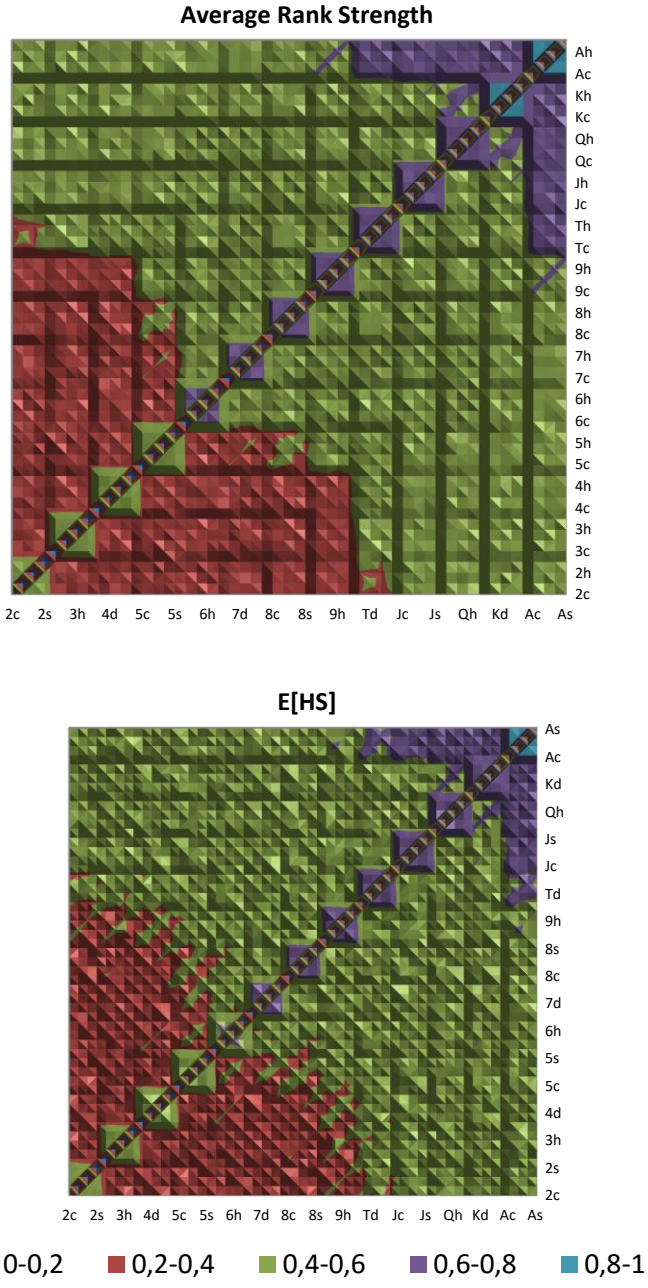


Figure 51 – Average rank strength VS E[HS] heat maps at River

5.3 Reduced Game Utility Abstraction (RGU)

In the first sections of this chapter, several methods were discussed (with a new one presented) that are used for Poker game abstraction. The discussed method *E[HS]* and its variations or improvements (*ARS*), are usually used for card abstraction. Since they return a percentage of how valuable is the hand (100% means that it is unbeatable), those intervals can be split and form card buckets (e.g. goods hands with *E[HS]* > 80%, mid hands with 80% >= *E[HS]* > 50% and bad hands with *E*[*HS*] <= 50%). However, this kind of intervals do not seem to fairly represent the game probabilities because they always assume that the opponents will not fold their hands, during the simulation. Moreover, this abstraction technique is Poker specific, which means that in scientific terms it is not as interesting as a more generic algorithm.

To overcome the limitations of the previous methods, a new methodology was created – **Reduced game utility abstraction.** The idea of this method comes from the concept of average utility or mathematical expectation – how much I will get from a certain action. This method considers games like Poker that have random pre-conditions that influence the flow of the game (in this case private cards). The idea is to do abstraction of the pre-conditions by using their average utility obtained by a set of Nash-Equilibrium strategies, which represent the utility of a solved game. Then, the utility values of each pre-condition could be used to group those conditions into buckets. For instance, in Poker the average utility of playing with A♠A♣ is much higher than the average utility of playing with 2♣4♠, but similar to the utility of playing with K♥K♣. Since the utility of a pair of Kings is similar to the utility of a pair of Aces, these hands can be grouped together in the same bucket.

The problem of the stated solution is that, for this to work, a Nash-Equilibrium strategy over the full unabstracted game is needed, which creates a recursive dependency. However, the Reduced game utility abstraction solves this problem by using a smaller similar game, with smaller sequences (in the case of Poker, smaller betting sequences).

We used the Limit Texas Hold'em game as example. Computing a Nash-Equilibrium strategy for this game is now possible but it requires a large amount of computational resources [24]. However, it is possible to easily compute a Nash-Equilibrium for unabstracted Leduc Poker, which is by far a much smaller game. Computing the Nash-Equilibrium for Leduc Poker with the full deck is still possible, because the number of game sequences is much lower than in Limit Texas Hold'em. After computing the Nash-Equilibrium strategies with CFR, the utility values are obtained either directly from the algorithm or they could be obtained by generating a high number of simulated games between the players of the Nash's strategy. The utilities can then be used to abstract the original large game. Figure 52 summarizes the steps to perform this approach.

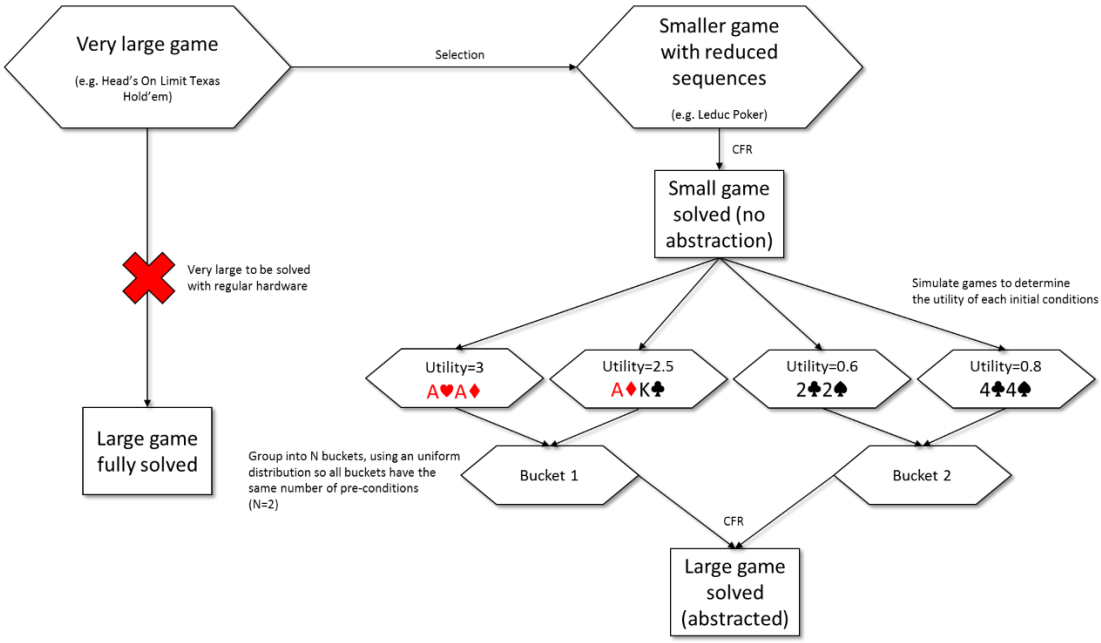


Figure 52 – Reduced game utility abstraction

RGU was validated against the *E[HS]* abstraction approach. Two Nash-equilibriums, for RGU and for *E[HS]*, were computed on Limit Texas Hold'em using Leduc Hold'em as the small game (with the full deck), with CFR and a training time of 2 hours each. Both abstraction approaches used uniform distributions to separate the N buckets. Then 1,000,000 matches were run between the agents produced by those two Nash-Equilibriums. The experience was repeated for different values of N. The obtained results are presented in Table 28 in total bankroll values.

Table 28 – *Reduced game utility abstraction tests in mili big blinds/h*

Number of buckets (N)	*E[HS]* abstraction	RGU abstraction
5	-9.41	9.41
10	0.52	-0.52
15	7.35	-7.35
20	-12.04	12.04
50	-15.08	15.08
100	-17.42	17.42
Average	**-7.68**	**7.68**

The obtained results show that statistically (with a small advantage for RGU) there are no major differences between both abstractions. However, since RGU is much more generic, its usefulness is demonstrated by these results even without having knowledge about the Strength of the Hands, the RGU abstraction still created a very competitive agent.

5.4 Summary

In this chapter two new abstraction methods were discussed and assessed. The **ARS** is a Poker specific method that represents an improvement of indexation in storing tables with *E[HS]*. The method is an approximation to *E[HS]* with a very low error, but very fast (1000 times faster) and uses 16 Mb of memory instead of 2.5 Gb. The second presented method, **RGU**, is an abstraction method based on the average utility of initial conditions computed by a Nash-Equilibrium set of strategies on a smaller game. It is thus a more general approach to abstraction.

Chapter 6

Game Playing

This chapter presents the developed algorithms or techniques to improve the game playing methodologies. The first sections present some techniques (not necessarily related to each other) that propose new ways to approach the development of competitive Poker agents (this includes improvements on the current state of the art technology for sequential games – the CFR algorithm). Sections 6.3 and 6.4 present respectively the Lucifer and Hermes architecture that are validated in Chapter 7.

6.1 Inferring Poker-Lang Strategies from Game Logs

6.1.1 Method Description

The first approach for building game playing algorithm during this thesis work was to try to imitate good Poker Players experts. One possible way of doing that is to have experts specifying *PokerLang* (see Section 3.2.5) documents. However, it takes a very long time to accurately describe a strategy with precision to achieve a good in game performance (even if they use the *PokerBuilder* interface). In order to surpass this problem, a new approach was designed to perform inference of rules from game logs – sets of recorded games G_P. This way, if any user has game playing data available, this new method will allow him or her to import a strategy from games of agents or humans that play with a strategy similar to the intended one. In summary, this new approach generates *PokerLang* documents from game data.

The built inferring system does not consider *PokerLang* predictors (information set features that are estimated from observable data); it just considers the following language evaluators: Stack: *St* (the amount of money that the copied player has in *the language* relative proportions: green, yellow, orange, red, dead); Hand Strength ranges *Hi* (see Section 3.5.1 for Hand Strength)*;* Position at table: *Po*. These features are information selectors which represent game conditions to activate a given strategy.

To build this system, we considered all possible combinations of these evaluators. However, since the hand strength is a continuous measure, its distribution has to be discretized. Let us analyse a distribution of hand strength values extracted from a particular collection of game logs[35] (Figure 53 and Figure 54 – The horizontal axis contains the values of hand strength (ranging from 0 to 1) and the vertical axis is the relative frequency of that hand strength value).

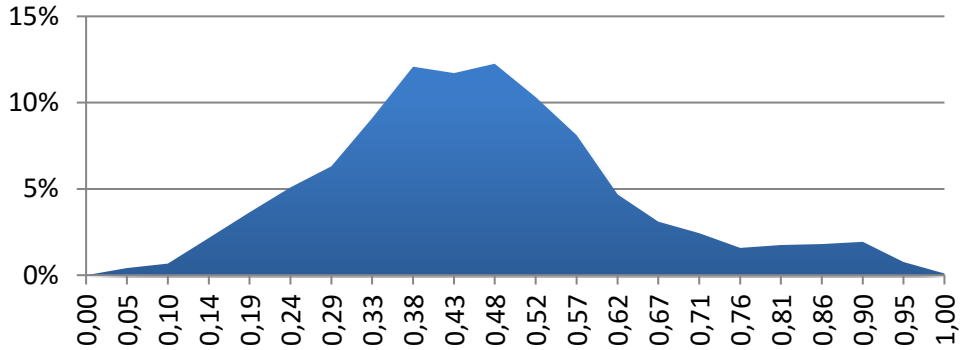


Figure 53 – Hand Strength relative distribution observed from the dataset in the Pre-Flop round.

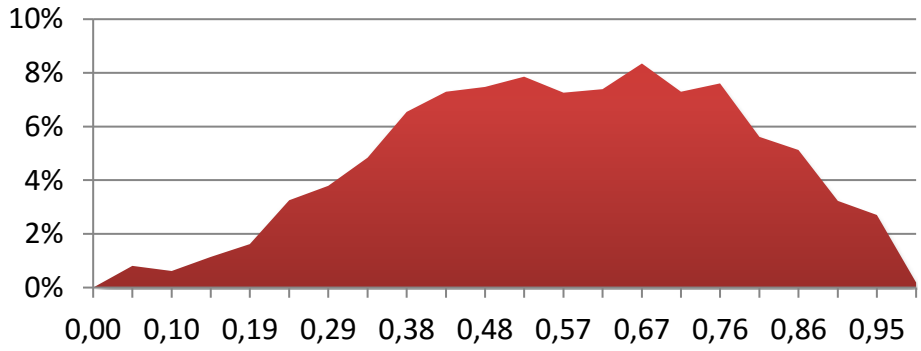


Figure 54 – Hand Strength relative distribution observed from the dataset in Post-Flop rounds.

[35] The collection of game logs was provided by a professional Poker player.

As expected, the frequency of high values of hand strength is higher on later rounds (Figure 54). This happens because the players successively give up weaker hands. Since the distributions are rather distinct, we differentiate them during the inferring process: when inferring evaluators in Pre-Flop rounds we use the distribution illustrated on Figure 53, and for other rounds we use the distribution visible in Figure 54.

The discretization process was simple: a fixed number of hand strength intervals (*k*). The interval offsets were chosen to obtain a uniform distribution based on the relative frequency of *HS* values.

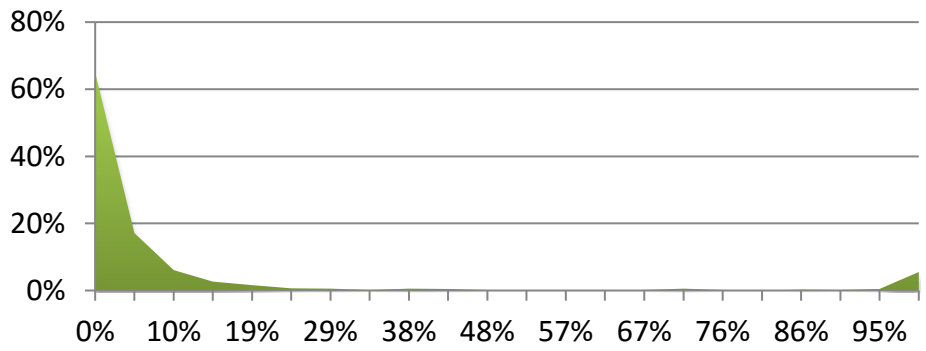


Figure 55 – Betting distributions for Pre-Flop round.

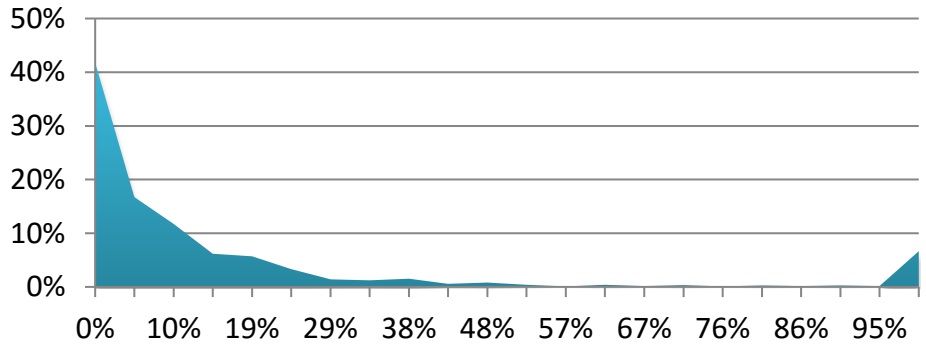


Figure 56 – Betting distributions for Post-Flop rounds.

A similar strategy was considered for the action output for the selectors – *Ad*. The betting distribution was also obtained from the game logs collections (Figure 55 and Figure 56 – the horizontal axis expresses the percentage of the player's money that was bet). After that, from the betting distribution a fixed number of intervals were extracted (*q*). Given this, the tuple that the inferring system must recognize is:

$$\langle St, Hi, Po, Ad\rangle \left| \begin{cases} St \in \{green, yellow, orange, red, dead\} \\ Po \in \{bad, normal, good\} \\ |Hi| = k \\ |Ad| = q \end{cases} \right.$$

EQ21

The number of recognizable tuples is given by $|St|\times|Hi|\times|Po|\times|Ad|=5\times k\times 3\times q$. In the experiments we arbitrarily used *k* = 10 and *q* = 10, making a total number of 1500 cases.

Three different approaches were tested to recognize a case from the game logs. The first one is a well-known classifier – the Random Forest Tree – that already proved empirically to be the best suited for Poker data [30]. The second strategy was to use the Euclidian distance between the extracted features and features from the static tuples – the closest case is the one to be activated. This was based on the methodology from [75] where two information sets have a degree of similarity equal to the average similarity of the game features. However, instead of the average, the degree of similarity was calculated as in [30] through the Euclidean distance between sets of features. Being *i* and *j* two information sets, $f \in F$ and f ∈ Fa the game features and i_f, j_f the values of feature *f* on those information sets, the distance is given by EQ22.

$$euclidean(i,j) \coloneqq \sqrt{\sum_{f}^{F} \left(i_f - j_f\right)^2}$$

EQ22

Finally, a modified version of the Euclidean distance was used – weighted Euclidean distance. The weighted Euclidian (EQ23) distance considers a weight vector w where w_f is the weight of feature *f*.

$$weighted\_euclidean(i,j) \coloneqq \sqrt{\sum_{f}^{F} w_f \times \left(i_f - j_f\right)^2}$$

EQ23

The weight vector was determined empirically by running the validation method described by EQ24. An agent that follows the inferred strategy was created and the accuracy for tuples with different weights was tested. The used weights greatly depend on the available data and on the copied player's strategy but, for instance, the *Hi* has usually a weight over 40%.

$$\mathrm{acc}(i, C, h, i^a) := \frac{|\{1: G_P \in C \wedge h \in G_P \wedge i \in G_P \wedge p(h) = i \wedge h_i^+ = h_i^+ a\}|}{|C|}$$

EQ24

In EQ24 *C* is the collection of cases for player *i*, h_i^+ is the history after the player *i* action and $\mathrm{h}_{\mathrm{i}}^{+}{}_{\mathrm{a}}$ is the action performed by the agent representing player *i*. The accuracy is the ratio between the number of cases where the agent selected an action similar to the player's original action and the total number of cases.

6.1.2 Weight selection and results

In experiments, to determine the weight vector, its weights are randomly generated so that $\sum_i^{|F|} w_i = 1$. Next, the agent is generated and its accuracy is determined for a fixed number of iterations. The agent with better accuracy is the one that it is selected by the system. Other policies can be used to determine the weights, namely genetic algorithms with populations of agents with different weight vectors. However, it is possible to check in Table 29 that the random generation policy already produced agents with very good accuracies.

The weighted Euclidian distance always produced agents with greater accuracy than the two other methods, with an average accuracy of ~79% for datasets with 5000 cases and 10.000 iterations, proving the usefulness of this method. In Table 29 logs of 10 different players were used. For each player, 3 sets of cases with different sizes were extracted (1000, 2500 and 5000). The game logs contained full game state description of the players from whom the strategies were inferred. The developed strategy inferring system proved empirically to be accurate for generating strategies similar to human ones from past played games.

Table 29 – *PokerLang* strategy inferring accuracy.

Method	*Random Forest*			*Euclidian Distance*			*Weighted Euclidian*		
Iterations	1000	2500	5000	1000	2500	5000	1000	2500	5000
Accuracy	38,1%	42,3%	41,6%	45,0%	52,2%	47,0%	55,0%	75,2%	80,1%
	25,8%	50,2%	63,8%	56,0%	57,6%	67,5%	65,5%	56,9%	70,5%
	50,6%	56,0%	68,7%	60,0%	66,1%	84,1%	50,5%	84,7%	86,4%
	45,6%	68,2%	67,5%	70,5%	71,1%	72,2%	53,8%	69,8%	73,2%
	30,2%	52,0%	56,4%	55,4%	64,5%	70,3%	47,4%	77,8%	81,6%
	56,6%	77,8%	78,6%	67,1%	76,3%	77,9%	67,5%	59,0%	79,3%
	50,6%	76,1%	75,7%	49,7%	51,3%	70,2%	45,1%	59,4%	78,4%
	62,4%	70,8%	82,1%	30,1%	66,0%	70,4%	33,6%	81,5%	86,9%
	33,3%	41,0%	50,9%	40,5%	65,6%	53,1%	51,0%	70,6%	75,4%
	61,3%	64,8%	67,1%	51,2%	67,9%	71,9%	54,5%	71,4%	79,1%
Average	45.5 ± 12.4 %	59.9 ± 12.8 %	65.3 ± 12.0 %	52.5 ± 11.5 %	63.9 ± 7.5 %	68.4 ± 10.3 %	52.4 ± 9.2 %	70.6 ± 9.2 %	79.1 ± 4.9 %

6.2 Optimizations on the CFR algorithm

6.2.1 A recursive implementation

The counterfactual regret minimization algorithm (CFR) as explained before is the current state of the art algorithm to solve very large sequential games, being far superior to linear programming, because it requires much less iterations – they are proportional to the number of information sets instead of the number of game states (which in Poker means at least 6 orders of magnitude less).

The CFR algorithm is a recursive algorithm, i.e. it transverses the game tree until it reaches the leaf nodes (in the case of Poker, nodes where the players show their cards to each other or nodes where the number of remaining players is 1). In this research work a regular recursive implementation was done to allow for the generation of Nash-Equilibrium strategies. The C++ implementation is presented in Figure 57. This implementation is generic and independent of the Poker variant and it uses the ACPC native C structures (this was done because this software was built to be an entry for the ACPC competition).

The recursivity of this regular CFR implementation can be seen in code line 40 (where the defined function ***calc*** calls itself). The end of the recursion is in line 5, where the algorithm verifies if the game state reached a final state or not.

```
01|void
02|CFR::calc(Game* game, State* state, double probs[], uint8_t
03| previousPlayer, double nodeUtil[])
04|{
05|    if(state->finished != 0) {
06|        for(int player = 0; player != game->numPlayers; ++player) {
07|            nodeUtil[player] = valueOfState(game, state, player);
08|        }
09|    } else {
10|        //get the information set
11|        uint8_t curPlayer = currentPlayer(game,state);
12|        std::string infoSet = abstraction(state);
13|        CFRNode* node = this->nodeMaps[curPlayer *
14|        this->game->numRounds + state->round].at(infoSet);
15|
16|        double strategy[game->numPlayers];
17|        node->getStrategy(probs[curPlayer], strategy);
18|
19|        memset(nodeUtil,0,game->numPlayers*sizeof(double));
20|        Action act;
21|
22|        double util[MAX_ABSTRACTED_ACTIONS];
23|
24|        for(int a = 0; a != MAX_ABSTRACTED_ACTIONS; ++a) {
25|            if(node->isActionValid(a)) {
26|                getAbstractedAction(a, state, &act);
27|                State nextState= *state;
28|                doAction(game,&act,&nextState);
29|
30|                double newProbs[game->numPlayers];
31|                for(int p = 0; p != game->numPlayers; ++p) {
32|                    if(p == curPlayer) {
33|                        newProbs[p] = probs[p] * strategy[a];
34|                    } else {
35|                        newProbs[p] = probs[p];
36|                    }
37|                }
38|
39|                double nextNodeUtilities[game->numPlayers];
40|                calc(game, &nextState, newProbs, curPlayer,
41|                  nextNodeUtilities);
42|
43|                util[a] = nextNodeUtilities[curPlayer];
44|
45|                for(int p = 0; p != game->numPlayers; ++p) {
46|                    nodeUtil[p] += strategy[a] *
47|                          nextNodeUtilities[p];
48|                }
49|            }
50|        }
51|        for (int a = 0; a < MAX_ABSTRACTED_ACTIONS; a++) {
52|            if(node->isActionValid(a)) {
53|                double regret = util[a] - nodeUtil[curPlayer];
54|                node->regretSum[a] += probs[previousPlayer] *
55|                                                    regret;
56|            }
57|        }
58|    }
59|}
```

Figure 57 – CFR recursive implementation for generic Poker variants

To better explain this implementation, it can be decomposed into the following parts:

- The information sets are stored in a dictionary that maps strings (concatenation of the private cards, community cards and game sequence) called **nodeMaps**. One particular detail about this implementation is that the round number is used (e.g. Texas Hold'em: 0 – PreFlop, 1 – Flop, 2 – Turn, 3 – River) to reduce the search for the information set probabilities, by separating it into 4 maps each of which containing the respective round's information sets.
- The algorithm checks if a final node was reached. In this case the node final utility is propagated (the players' cash prizes) to its parent nodes, by filling the **nodeUtil** array (lines 5-8)
- The algorithm tries to perform all actions over the current game state (line 24) if they are valid (line 25)
- After getting the action's utility (line 28), the strategy probabilities are updated for the current node (lines 30-49)
- Finally, the counterfactual regret is updated for all actions (see lines 51-57)

To run the algorithm it is only necessary to do several iterations with it by simulating random games. The more iterations are done, the more the algorithm will possibly be closer to a Nash-Equilibrium unless overfitting happens, which is a rare event [80].

6.2.2 A new proposed solution – an iterative implementation of CFR

One of the problems of a CFR implementation like the one that was described above (Figure 57) is that it needs a huge amount of iterations – this means that a lot of full tree traversals (one per simulation) must be performed and this is a problem especially on the leaf nodes where recursivity can greatly increase the heap size due to the large number of possible game sequences in Texas Hold'em. In order to overcome this problem, one proposed solution (that to the best of the author's knowledge was not tried before) is an iterative implementation of the algorithm. Since CFR works with *backpropagation* of utilities, turning CFR into an iterative algorithm requires it to

process each level of the tree independently, from the deepest leaf node to the root node. The goal of this new implementation was to verify if it was faster than the original CFR (current implementations can take several days of computation even in very powerful computers).

This approach has very good advantages with the main one being the possibility of using parallelization without error penalties on the algorithm and without any need for a synchronization mechanism. Current parallelized versions of CFR just train the algorithm with several games at the same time – this simultaneous training can spoil some information sets that can be updated concurrently (especially when running on abstracted versions). Using semaphores to avoid collisions would solve this issue but it would make the algorithm even slower. Another advantage is that an iterative version of CFR would allow for using the GPU capabilities for concurrent operations. With linear arrays instead of tree structures, it is possible to create a GPU version of CFR that could benefit from high parallelization. The idea of using a GPU is also good because most CFR's operations are arithmetical – GPUs are known to be very fast to perform this kind of operations. This was done later as an extension of this work in [81].

One potential disadvantage of an iterative CFR is information representation. Representing game trees in a linearized way generates very sparse arrays that occupy a lot of memory that is actually not used. For instance, to represent the 2 player's Kuhn Poker variant game tree and the correspondent array that is used for storing the information (Figure 58 – consider the actions to the left a Call / Fold and the actions to the right a Raise). In this very small game (with only four levels of depth) there are 6 unoccupied positions, meaning that 40% of the space is wasted – positions {7, 8, 11, 12, 13, 14}. This is the biggest potential disadvantage of this approach. However, this could be partially solved with efficient sparse array representations such as hash maps. Nevertheless, one could argue that, in the last years, memory became much cheaper which allows for having a little prejudice on the amount of used memory in order to speed up the algorithms.

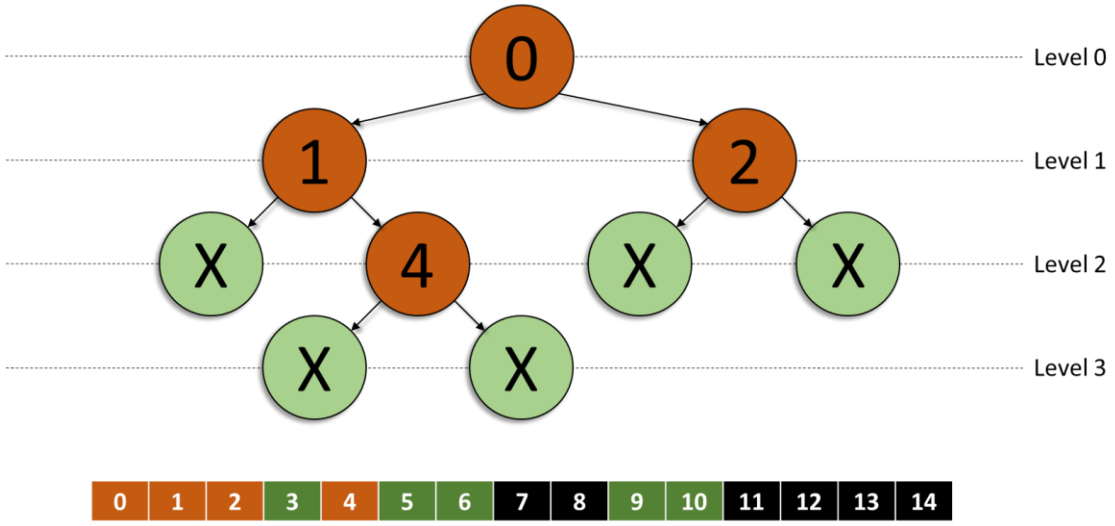


Figure 58 – Kuhn Poker's strategy into sparse arrays.

6.2.2.1 Implementation

In order to implement a non-recursive CFR algorithm, it is necessary to represent all required variables in plain arrays, in order to store the data created by the algorithm's recursivity. Referring to the implementation in Figure 57, the variables that need to be linearized are (for small games the linearization was done like in Figure 58 for simplification purposes, without supporting data structures):

- **RegretSum** – the accumulated values of regret for a node
- **StrategySum** – the accumulated sum of all strategy values i.e. the output of the algorithm for all game nodes (processed on getStrategy function in Figure 59 and returned by the **calc** function on Figure 57)
- **Average Strategy** (strategy) – the used strategy values for the current iteration
- **Node Utility** (nodeUtil) – the node utility for the current iteration
- **Probabilities of Information sets** (probs) – an array that contains for each node the probability of it being reached.

To implement this approach, the order of the algorithm steps must be changed:

- Update the probabilities (probs) and average strategy by levels – starting at the root level
- Update the counterfactual regret and the node utilities in the reverse order – from the last level nodes to the top of the tree.

— The parallelism can be applied only to nodes of the same level, i.e. it only justifies dividing the work in levels where the tree is very large. Doing parallelization on the first levels would present a very high overhead for a small set of calculations

```
01|CFRNode::getStrategy(double realizationWeight, double
02|strategy[MAX_ABSTRACTED_ACTIONS])
03|{
04|  double normalizingSum = 0;
05|  for (int a = 0; a < MAX_ABSTRACTED_ACTIONS; a++) {
06|    if(isActionValid(a)) {
07|      strategy[a] = regretSum[a] > 0 ? regretSum[a] : 0;
08|      normalizingSum += strategy[a];
09|    } else {
10|      strategy[a] = 0;
11|    }
12|  }
13|  if(normalizingSum > 0) {
14|    for (int a = 0; a < MAX_ABSTRACTED_ACTIONS; a++) {
15|      if(isActionValid(a)) {
16|        strategy[a] /= normalizingSum;
17|        strategySum[a] += realizationWeight * strategy[a];
18|      }
19|    }
20|  } else {
21|    double prob = 1.0 / getNumValidActions();
22|    for(int a = 0; a < MAX_ABSTRACTED_ACTIONS; a++) {
23|      if(isActionValid(a)) {
24|        strategy[a] = prob;
25|        strategySum[a] += realizationWeight * strategy[a];
26|      }
27|    }
28|  }
29|}
```

Figure 59 – GetStrategy function (CFR implementation) is the function that updates the actual strategy probabilities taking into account the current accumulated regrets.

One important thing is to determine inside the linearized trees (arrays) which nodes are the ones that belong to that level. That can be done by ϑ_{min} and ϑ_{max} described on EQ25. *MAX_ABSTRACTED_ACTIONS* represent the maximum number of actions in the variant that is being processed (in Kuhn / Leduc Poker this value is 2, in Limit Hold'em this value is 3 and in No Limit Texas Hold'em this value depends on the abstraction).

$$\vartheta_{min}(0) = 0, and\ \vartheta_{max}(0) = 0$$

$$\vartheta_{min}(level) = \vartheta_{max}(level - 1) + 1, level > 0$$

$$\vartheta_{max}(level) = \vartheta_{max}(level - 1) + 3^{Level}, level > 0$$

EQ25

These values are pre-computed since their definition is recursive, in order to reduce the number of calculations.

The algorithm in Figure 60 summarizes the described steps for the final strategies computation. All the algorithm parameters are trees represented in linearized arrays like was explained in Figure 58.

Algorithm $IterativeCFR(RegretSum, AvgStrategy, NodeUtils, Prob, CanDoAct)$

```
Let NLevels := 0
Let CurLen := Length(RegretSum) + 2
While CurLen > 1
  CurLen = CurLen / 2
  NLevels = NLevels + 1
For Level := 0 to NLevels-1
  For Index := ϑ_min(Level) to ϑ_max(level)
    strategy := getStrategy(RegretSum[index], AvgStrategy[index])
    updateAvgStrategy(AvgStrategy[Index], Prob[index])
    For Action := 0 to MaxActions
      If CanDoAct[Index]
        updateProbs(Prob[Index])
For Level := NLevels – 1 to 0
    For Index := ϑ_min(Level) to ϑ_max(level)
      For Action := 0 to MaxActions
          childIndex := ϑ_max(level + 1) + MaxActions² × ((Index + Action − ϑ_min(Level)) ÷ MaxActions)/)
          nodeUtils[index] = nodeUtils[childIndex] * avgStrategy[Index][Action]
      updateCfrRegret(nodeUtils[Index], RegretSum[Index], Prob[Index])
return
```

Figure 60 – Liner CFR algorithm

Some notes about this implementation:

- *CanDoAct* is an array of Booleans that marks all information sets as being possible or not (because of the sparse array problem described earlier in this section)
- *getStrategy* – a function that returns the current strategy (see Figure 59)
- *updateAvgStrategy*, *updateProbs* and *updateCfrRegret* – all refer to the methods described in Figure 57.

6.2.2.2 Results

In order to test the new iterative approach for the CFR algorithm, several Nash-Equilibrium strategies were computed with both the recursive and the iterative versions of CFR, for Poker variants of different size. The results for respectively speed

and memory efficiency are presented below in Table 30 and Table 31 (10.000 iterations were run without any abstraction). Tests on Limit Texas Hold'em or any Hold'em variant were not performed because of the high memory requirements and it is important to analyze the results without abstraction because they can potently change the depth of the game.

Table 30 – Recursive CFR vs Linear CFR (time in seconds)

Game	Recursive CFR	Iterative CFR	Difference (%)
2P Kuhn (16 information sets)	0,08	0,25	-212,50%
2P Kuhn, Full Deck (33.390.720 information sets)	0,22	0,42	-90,91%
5P Kuhn, Half-Deck (43.080.840 information sets)	382,15	180,61	**52,74%**
8P Kuhn, Quarter-Deck (4.932.736 information sets)	14.497,04	4.939,55	**65,93%**

Table 31 – Recursive CFR vs Linear CFR (memory usage in MB)

Game	Recursive CFR	Iterative CFR	Difference (%)
2P Kuhn (16 information sets)	1,32	1,56	-18,18%
2P Kuhn, Full Deck (33.390.720 information sets)	1835,82	1836,00	-0,01%
5P Kuhn, Half-Deck (43.080.840 information sets)	992,43	1046,19	-5,42%
8P Kuhn, Quarter-Deck (4.932.736 information sets)	201,71	1650,30	-718,15%

From Table 30 it can be observed that the time reduction is very high when the variant is big enough (reduction of more than 50%). However, when the variant is small, the time spent even increases – probably due to the overheads of preparing and loading the game tree to memory. However, the penalty on memory usage when using

an iterative version of the CFR could be huge, with memory usage increases of more than 700% in the deepest game (we had memory increases in all tests, according to Table 31). Despite this last test, these results prove the usefulness of this approach, when a large amount of memory is available or when the game tree is not too deep. With improvements on the sparse arrays storage, this memory increase will certainly reduce (with a speed penalty).

6.2.3 Pruning the CFR search tree

One common procedure before using the CFR algorithm is to create and store the game tree before running it – this increases the efficiency of the algorithm because it now can assume that all information sets exist (it does not need to verify them every time). For this CFR implementation, the used approach is based on the code on Figure 60. This approach slightly reduces the number of processed game nodes by considering some possible actions as impossible actions. This refers to not loading nodes that cannot be reached by considering completely unwise actions as impossible actions (e.g. folding a hand instead of doing a call when no money has to be spent – see line 23 and line 24 in Figure 61, where the action fold is removed when the money spent by the player is equal to the max amount that any player has spent so far).

Although useful, this approach does not allow for removing a lot of game nodes – only about 0.5% of the nodes. In order to increment the algorithm's efficiency even further, a new method was developed based on the concept of strategic dominance. A dominance occurs when, no matter what, a given player's action will result in a win. One perfect example is when a player has to decide to Call or Fold, when holding a Royal Flush in No-Limit Texas Hold'em and all his or her opponents are in All-In state. The player does not know the cards of his or her opponents, but in this case it does not matter – he or she will win no matter what the opponents are holding – this is called a dominant play. Another way around is being in a very similar situation, but holding the worst possible hand. If the player calls, he or she will lose the game for sure – this is called a dominated play.

```
01|void
02|CFR::createAbstractedGameTree(State* state)
03|{
04|  if(state->finished != 0) {
05|    return;
06|  } else {
07|    std::string infoSet = abstraction(state);
08|    uint8_t curPlayer = currentPlayer(game,state);
09|    if(this->nodeMaps[curPlayer * this->game->numRounds +
10|           state->round].count(infoSet) == 0) {
11|      this->nodeMaps[curPlayer * this->game->numRounds +
12|      state->round].insert(std::pair<std::string,
13|           CFRNode*>(infoSet, new CFRNode()));
14|    }
15|    CFRNode* node = this->nodeMaps[curPlayer *
16|         this->game->numRounds + state->round].at(infoSet);
17|
18|    bool validActions[MAX_ABSTRACTED_ACTIONS];
19|    Action act;
20|    for(int action = 0; action != MAX_ABSTRACTED_ACTIONS;
21|         ++action){
22|      getAbstractedAction(action,state,&act);
23|      if(act.type == a_invalid || (act.type == a_fold &&
24|           state->spent[curPlayer] == state->maxSpent)) {
25|        validActions[action] = false;
26|        continue;
27|      }
28|
29|      State nextState = *state;
30|
31|      if(isValidAction(game,&nextState,0,&act)) {
32|        doAction(game, &act, &nextState);
33|        createAbstractedGameTree(&nextState);
34|        validActions[action] = true;
35|      } else {
36|        validActions[action] = false;
37|      }
38|    }
39|    node->initialize(validActions);
40|  }
41|}
```

Figure 61 – Building the CFR actions tree (C++)

The new developed method is based on the described concepts (dominant and dominated actions) and is comprehended in Figure 61. The idea is to consider almost dominant actions as dominant and almost dominated actions as dominated, using the winning probability as measure to do that. In order to allow for parameterizing and adapting the algorithm to several different situations, two parameters were included: *MAX_WIN_PROB_THRESHOLD* and *MIN_WIN_PROB_THRESHOLD*. These refer respectively to the minimum and maximum value of winning probability that will be considered dominant and dominated play. For instance, having MAX_WIN_PROB_THRESHOLD=5% means that any hand with less than 5% probability of winning will automatically be considered a dominated play and, therefore, all subsequent game nodes will be removed. By using the thresholds (5%, 95%) it is possible to reduce the tree length by about 8%, with minimum impact.

```
01|void
02|CFR::eliminateActions() {
03|    std::string idStr;
04|    double prob;
05|    unsigned long long count = 0;
06|    for(int i = 0; i != (game->numPlayers*game->numRounds); ++i) {
07|        for(std::map<std::string, CFRNode*>::iterator it =
08|             nodeMaps[i].begin(); it != nodeMaps[i].end(); ++it) {
09|            idStr = it->first.substr(4);
10|            idStr = idStr.substr(0,idStr.find(':'));
11|            prob = winProb(atoi(idStr.c_str()));
12|            if(prob >= MAX_WIN_PROB_THRESHOLD) {
13|                it->second->setActionInvalid(a_fold);
14|                count++;
15|            }
16|        }
17|    }
18|    for(int i = 0; i != (game->numPlayers*game->numRounds); ++i) {
19|        for(std::map<std::string, CFRNode*>::iterator it =
20|             nodeMaps[i].begin(); it != nodeMaps[i].end(); ++it) {
21|            idStr = it->first.substr(2);
22|            idStr = idStr.substr(0,idStr.find(':'));
23|            int round = atoi(idStr.c_str());
24|            if(round == (game->numRounds - 1)) {
25|                bool anyRaiseAvailable = false;
26|                for(int a = a_raise; a < MAX_ABSTRACTED_ACTIONS;
27|                     ++a) {
28|                    if(it->second->isActionValid(a)) {
29|                        anyRaiseAvailable = true;
30|                        break;
31|                    }
32|                }
33|                if(!anyRaiseAvailable) {
34|                    idStr = it->first.substr(4);
35|                    idStr = idStr.substr(0,idStr.find(':'));
36|                    prob = winProb(atoi(idStr.c_str()));
37|                    if(prob <= MIN_WIN_PROB_THRESHOLD) {
38|                        it->second->setActionInvalid(a_call);
39|                        count++;
40|                    }
41|                }
42|            }
43|        }
44|    }
45|    if(AGENT_RANGE < 1.0) {
46|        for(int i = 0; i != (game->numPlayers*game->numRounds);
47|             ++i) {
48|            for(std::map<std::string, CFRNode*>::iterator it =
49|              nodeMaps[i].begin(); it != nodeMaps[i].end(); ++it) {
50|                idStr = it->first.substr(4);
51|                idStr = idStr.substr(0,idStr.find(':'));
52|                prob = winProb(atoi(idStr.c_str()));
53|                if(prob <= (1-AGENT_RANGE)) {
54|                    for(int a = a_raise; a <
55|                         MAX_ABSTRACTED_ACTIONS; ++a) {
56|                        it->second->setActionInvalid(a);
57|                    }
58|                    if(it->second->isActionValid(a_fold) &&
59|                          it->second->isActionValid(a_call)) {
60|                        it->second->setActionInvalid(a_call);
61|                    }
62|                }
63|            }
64|        }
65|    }
66|}
```

Figure 62 – Eliminating search nodes based on actions dominance (C++)

6.3 The ACPC Participation – Lucifer Agent Architecture

In this section the methodology that was followed to implement the Lucifer agent is demonstrated as well as the ***K-Current-Best-Utility*** method. The Lucifer agent was made especially to participate in the 2014 ACPC competition, in the multiplayer Kuhn Poker track, a very simple variant for 3 players, 1 round and 4 playing cards (Jack, Queen, King and Ace). Rules and details about this competition track and results can be found in Section 7.2.

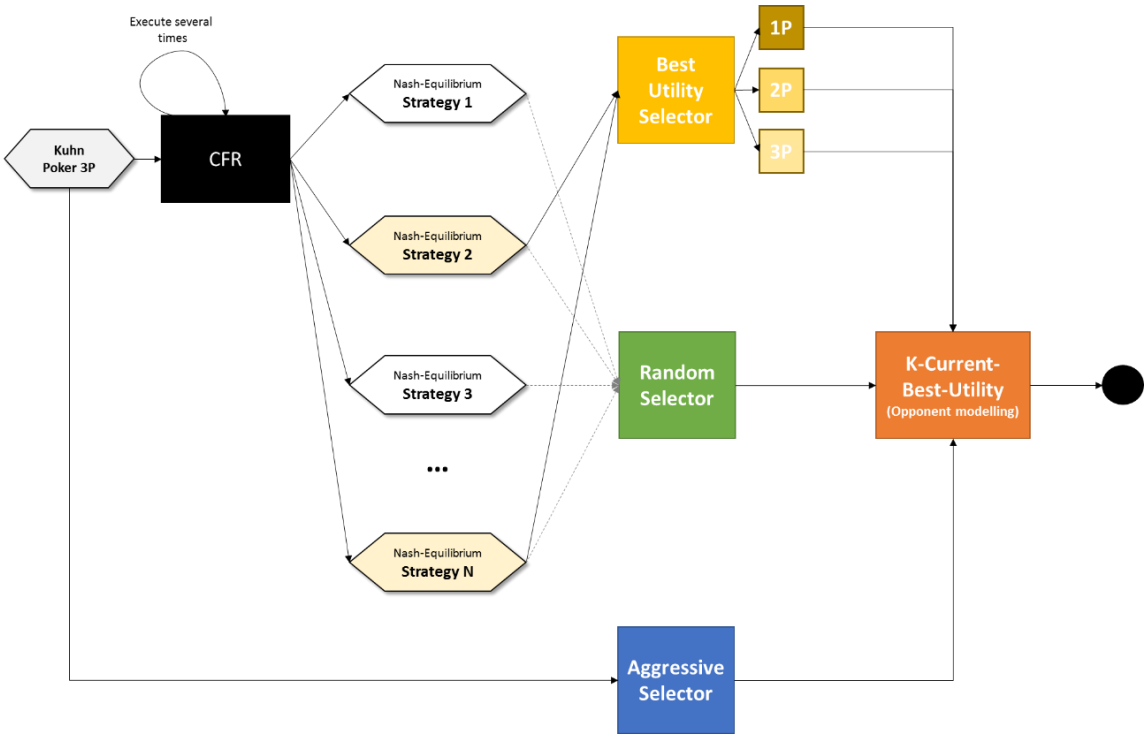


Figure 63 – Lucifer's Architecture

The Lucifer's global architecture is depicted in Figure 63 and the main parts of the code in Figure 64. The agent's architecture can be essentially divided into 3 parts:

— **CFR** – the linear implementation described in 6.2 was used. Using the iterative version, despite its usefulness, would not benefit the agent because the 3 player Kuhn game is too small. Several Nash-Equilibrium strategies are computed (in this particular case 1.000), with 1.000.000 iterations for training. All strategies are previously computed, so the CFR algorithm is not directly used by the agent – only the just strategies generated from it are used (see CFR usage between lines 57 and 67 in Figure 64).

- **Selectors** – these modules of the application are responsible for selecting possible actions for the algorithms actuator. In sum, they represent the agent's strategies. There are 3 types:
  - **Best Utility –** this strategy selects the Nash-Equilibrium strategies for each position in the table (in this case 3, because there are 3 players). The selection of strategies is based on the average utility for the position obtained by CFR. So, the algorithm selects the strategy that maximizes utility for that position (see strategy selection between lines 73 and 83, when the agent receives the hole cards; in line 62 the condition selects the CFR strategy for the player; in the *getAction* function, lines 87-93, the agent executes a **best utility** move).
  - **Random** – this strategy just selects one of the random and pre-computed Nash-Equilibrium strategies to play. See lines 93-98.
  - **Aggressive** – this a very simple strategy without Equilibrium concept. It always raises when the agent has the top 2 cards (King and Ace), otherwise it Folds or Calls (if it is a free call). See lines 99-114.
- **Opponent Modelling** – an opponent modelling module that uses the **K-Current-Best-Utility** for deciding which strategy is going to be used from the selectors.

Several functions of the Lucifer agent are presented in Figure 64, which represent the main parts of the agent's gameplay:

- **getEv** – this method is used by the K-Current-Best-Utility for computing the strategy's current utility;
- **updateEv** – updates the utility of the current selected strategy. It is called when a game ends and it uses its result to update the utility.
- **holeCards** – the event when the agent receives the cards. Here the agent just selects the current strategy.
- **getAction** – it contains the code for the agent to perform the action, from the currently selected strategy.

```
001|double Lucifer::getEv(int stra)
002|{
003|    double* arr;
004|    int len;
005|    if(stra == 0) {
006|        arr=profit_nash;
007|        len=len_nash;
008|    } else if(stra == 1) {
009|        arr=profit_br;
010|        len=len_br;
011|    } else if(stra == 2) {
012|        arr=profit_agressive;
013|        len=len_agressive;
014|    }
015|    double ev = 0;
016|    for(int i = 0; i != RECALL_SIZE; ++i) {
017|        ev+=arr[i];
018|    }
019|    return ev/RECALL_SIZE;
020|}
021|
022|void Lucifer::updateEv(int stra, double ev)
023|{
024|    double* arr; int* len;
025|    if(stra == 0) {
026|        arr=profit_nash;
027|        len=&len_nash;
028|    } else if(stra == 1) {
029|        arr=profit_br;
030|        len=&len_br;
031|    } else if(stra == 2) {
032|        arr=profit_agressive;
033|        len=&len_agressive;
034|    }
035|    if(*len == RECALL_SIZE) {
036|        *len = 0;
037|    }
038|    arr[*len] = ev;
039|    (*len) += 1;
040|}
041|
042|Lucifer::Lucifer() : PokerAgent()
043|{
044|    for(int i = 0; i != RECALL_SIZE; ++i) {
045|        profit_nash[i] = 0;
046|        profit_br[i]=0;
047|        profit_agressive[i]=0;
048|    }
049|    len_nash=0;
050|    len_br=0;
051|    len_agressive=0;
052|    double curUtil[game->numPlayers];
053|    double maxUtil[game->numPlayers];
054|    for(int i = 0; i != game->numPlayers; ++i) {
055|        maxUtil[i] = -100000.0;
056|    }
057|    CFR* cfr = new CFR(game);
058|    for(int i = 0; i != FIND_EQUILIBRIUM_ITERATIONS; ++i) {
059|        cfr->train(1000000);
060|        cfr->calcUtility(curUtil);
061|        for(int j = 0; j != game->numPlayers; ++j) {
062|            if(curUtil[j] > maxUtil[j]) {
063|                maxUtil[j] = curUtil[j];
064|                this->cfr[j] = cfr;
065|            }
066|        }
067|    }
068|    for(int i = 0; i != MAX_KUHN_OPPONENTS; ++i) {
069|        this->cfr[i]->calcUtility(curUtil);
070|    }
071|}
072|
073|void Lucifer::holeCards(uint8_t* holeCards, uint8_t seat)
074|{
075|    double maxEv = -1000;
076|    for(int i = 0; i != 3; ++i) {
077|        double ev = getEv(i);
```

```
078|         if(ev > maxEv) {
079|             currentStrategy = i;
080|             maxEv = ev;
081|         }
082|     }
083|}
084|
085|void Lucifer::getAction(Action& action)
086|{
087|     if(currentStrategy == 0) { //best nash
088|         CFRNode* node = cfr[state.viewingPlayer]->findNode(&this-
089|>state.state);
090|         int index = node->getRandomActionIndex();
091|         cfr[state.viewingPlayer]->translate(index, &this-
092|>state.state, action);
093|     } else if(currentStrategy == 1) { //random nash
094|         CFRNode* node = cfr[pick_a_number(0,2)]->findNode(&this-
095|>state.state);
096|         int index = node->getRandomActionIndex();
097|         cfr[state.viewingPlayer]->translate(index, &this-
098|>state.state, action);
099|     } else { //agressive
100|
101|if(rankOfCard(state.state.holeCards[state.viewingPlayer][0])>=2) {
102|             action.type = a_raise;
103|             action.size = 0;
104|         } else {
105|             if(state.state.maxSpent ==
106|state.state.spent[state.viewingPlayer]) { //free call
107|                 action.type = a_call;
108|                 action.size = 0;
109|             } else {
110|                 action.type = a_fold;
111|                 action.size = 0;
112|             }
113|         }
114|     }
115|}
116|
117|void Lucifer::gameOverEvent(double payoff)
118|{
119|     updateEv(currentStrategy, payoff);
120|}
```

Figure 64 – Main parts of Lucifer's source code (C++)

The **K-Current-Best-Utility** is the opponent modelling methodology that Lucifer uses (see code above). It consists on selecting a strategy among several that has the currently higher average utility. Selecting a strategy that has more utility against an opponent is a plain choice, however this does not consider that the opponent might change strategy or that the model could have been simply miscalculated. In order to adapt to possible opponent strategy changes, this selection method also has a recall value, i.e., the maximum number of games where we can store utility. If the value of **K** is exceeded, the older utility values are forgotten (see Figure 65 for an example; for K=3, the selected strategy is the last one, but if we had K=5 the selected strategy would be the second one).

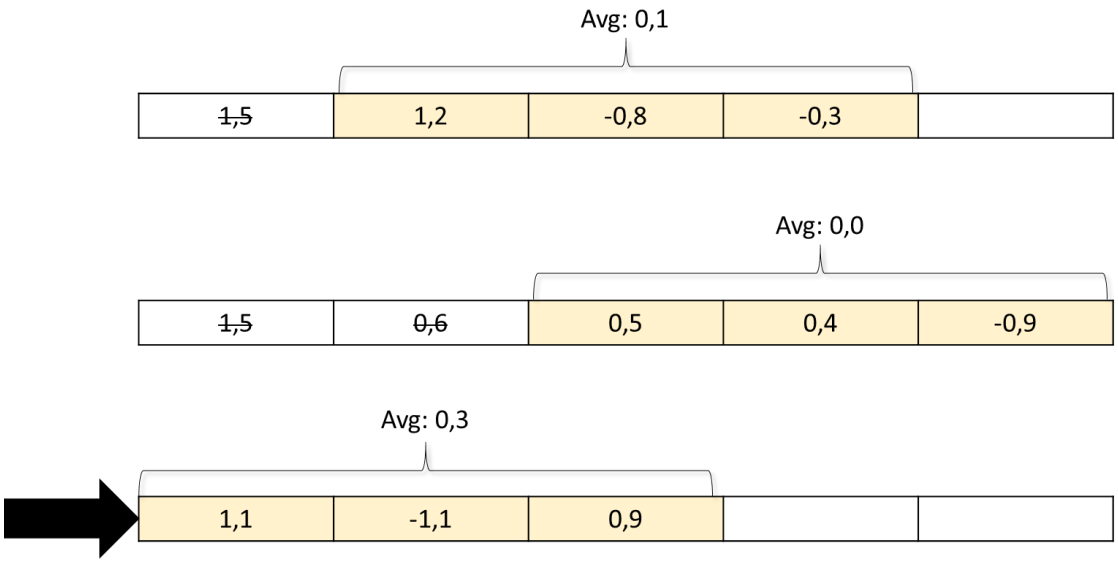


Figure 65 – K-Current-Best-Utility strategy selection example for K=3

By default, if not enough utilities were computed yet, 0 utility is considered (the utilities array starts with **K** zeros) – the utility that is stored is the profit on a given game. This means that the first strategy (**Best Utility**) only changes when the average utility is below 0, i.e. if by chance the strategy only made profit for groups of **K** games, it would never be changed. The **K** value that was used by Lucifer in the ACPC competition was 10. The determination method was merely empiric – i.e. several simulations were done against other agents (random agents, aggressive agents and Nash-Equilibrium agents) and best results were obtained with **K** = 10.

6.4 Online Game Playing – Hermes Agent Architecture

In this section the methodology that was followed to implement the online game playing agent named Hermes is demonstrated. The development approach was divided in three phases:

Online room interface – an interface which allows for Poker playing agents to impersonate a human player. In other words, this interface recognizes what is going on in a Poker room, provides the information to the software agents, receives the agent's response and finally controls the mouse and the keyboard to play accordingly to the agent's desire (see Section 4.3 for details).

Extracting opponent models – this consists of observing the opponents actions and label each one with a strategy type. The action of our agent's strategy depends on the types of strategies of the current opponents. An external tool called Hold'em Manager[36] was used for support in this phase.

The agent's strategy, which is based on a rule-based strategy from an expert player. This module is completely independent of the aforementioned, i.e. the agent can provide outputs and receive inputs from different platforms. This allows for testing the agent in a simulation environment, against other previously developed agents, without much extra effort. This was important to reduce the tests costs because the performed experiments with this agent were online in real money games (see Section 7.1).

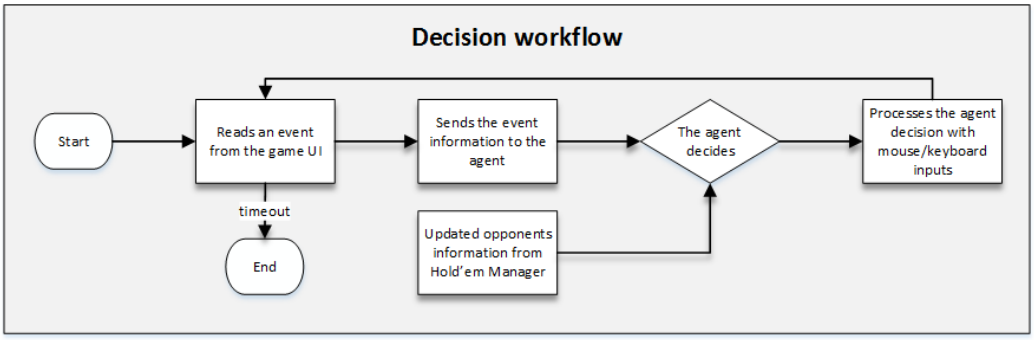


Figure 66 – Hermes's decision workflow.

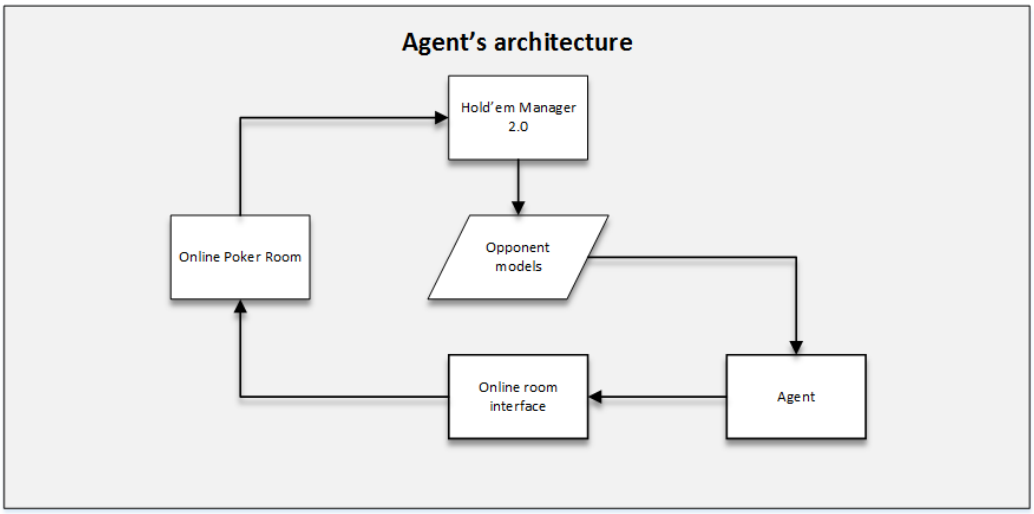


Figure 67 – Hermes's architecture.

[36] Hold'em manager website: http://www.holdemmanager.com/

The diagrams in Figure 66 and Figure 67 summarize the global view of the agent and how the different components communicate. The decision workflow is an endless cycle, i.e. the agent keeps reading events from the table. The cycle is interrupted when the agent is unable to read from the Poker Game UI which causes a timeout in the *"Read an event from the game UI"*.

6.4.1 Extracting opponent models

The opponent models are based on three common statistics about the players (VPIP, Fold3Bet and PFR). These statistics are collected during the games. The more the agent plays against a certain player, the more these statistics will reflect the opponents' playing style.

- $vpip(opponent \in N)$ – This statistic value stands for "Voluntarily Put $ In Pot" and tells the percentage of times a player makes a call or a raise on pre-flop round.
- $fold3bet(opponent \in N)$ – This statistic value tells the percentage of times a player folds the hand when one of its opponents raise at least two times in the same round. That value will be useful to calculate if the expected return is positive or negative against the hand the agent holds.
- $pfr(opponent \in N)$ – This statistic value tells the percentage of times a player raises a hand on the Pre-Flop round.

All these statistics are computed automatically by the Hold'em Manager software and are stored in a relational database. The agent extracts these through a direct connection to the Hold'em Manager's database.

6.4.2 The agent's strategy

Let's consider $hermes \in N$ being the developed agent playing a particular G_P. The developed agent follows a short-stack strategy. A short stack strategy has the following characteristics:

- Playing with a money stack (money brought to the game) of at most 20 big-blinds (minimum bet value).$\forall h: s(hermes, h) + b(hermes, h) \leq 20 \times c(h_0)$, being $h0$ the history of the first game decision.
- Initial number of opponents between 4 and 6. $5 \geq |N| \geq 7$. One of the conditions for a short-stack strategy to work well is the restriction of the number of players. When this condition is not met, the wiser decision is to exit that game and enter in another.
- Decisions are limited to the Pre-Flop round, knowing that $|S_{flop}| = \emptyset$, which means that the decisions only consider the Hermes's private cards.
- Hermes' decision abstraction. Hermes only chooses from three possible actions – fold, call and all-in – ignoring all possible raise values. The call action is only used if the Hermes decides to fold when the call action is free. In short, for a given history *h* where $p(h) = hermes$ (it is hermes' turn) then $a(h) \in \{0, s(p(h), h)\}$.

Before describing the algorithm, it is important to describe how to compute the equity (Algorithm in Figure 68). The equity is the probability of a certain player's hand winning when dealing the remaining hidden shared cards. It is similar to *E*[*HS*] or *ARS* (see Section 5.2.3) but it considers more carefully possible opponents' decisions.

Since Hermes is only making Pre-Flop decisions, there are no visible shared cards which means that we have to sample possible shared cards (with Monte Carlo simulation). The same happens for opponents' cards, because they remain hidden the whole game (and they might not even be revealed at all). For the opponent card sampling, a new variable $Perc$ is introduced as input (and here resides the main difference of this method to the *E*[*HS*]). $Perc$ indicates the percentile of the strength of possible opponents' starting hands. For instance, if $Perc = 28\%$, it means that we consider that our opponent is only likely to have the best 28% starting hands. This percentage reflects the hands' strength on the Pre-Flop, because Hermes only plays on the Pre-Flop. This means that Hermes never considers how the opponents' strategies work after the Flop.

Algorithm $Equity(h \in H, hero \in N, perc, niter)$

$win = 0$
$tie = 0$
$lose = 0$
$iter = 0$

Let $pairs$ = the list of the possible card pairs, ordered by value
$pp = sublist(pairs, (1 - perc) \times len(pairs), len(pairs))$

for each p **in** $pp \backslash P_{hero}$
  while $iter < niter$ **do**
    $Let\ board = gen\_random\_board(D \backslash p \backslash P_{hero}, 5)$
    $ourrank = \max_{w \in [P_{hero} \cup board]^5} score(w)$
    $opprank = \max_{w \in [p \cup board]^5} score(w)$
    if $ourrank > opprank$ **then** win++
    else if $ourrank < opprank$ **then** $lose$++
    else $tied$++
    end if
    $iter$++
  end while
end for each

return $\left(1 - \frac{lose}{win+tie+lose}\right)$

Figure 68 – Hermes equity computation algorithm

The next step is to evaluate the game state. The game state evaluation considers the number of players that have called ($\#callers$), the number of players that have raised ($\#raisers$) and the number of players that are all-in ($\#alliners$). Table 32 indicates the possible abstracted game states.

Table 32 – ***Possible game state abstractions considered by Hermes***

State	$\#callers$	$\#raisers$	$\#alliners$
unopened	0	0	0
limped	1	0	0
raised	0	1	0
allin	0	0	1
limps	>1	0	0

Next, we need to classify the Hermes' starting hand strength. For this, we need two measures: the hand classification function $hclass: D^2 \rightarrow \{1,2,3,4,5,6,7,8\}$, given by Table 33 and the expected hand return given by algorithm in Figure 69.

Algorithm $ExpectedReturn(Hermes \in N, opponent \in N, h \in H)$

$Let\ f3b = fold3bet(opponent)$

$Let\ bbs = vpip(opponent)$

$Let\ eq = Equity(h, hero, bbs, 10000)$

$Let\ h_0$ be the prefix of h where $|h_0| = 0$

$Let\ pot = \sum_{i}^{N} b(i,h)$

return $\left(\begin{pmatrix}(f3b - |N| \times c(h_0) \times pot) + \\ ((1 - f3b) \times (eq) \times (bbs + pot))\end{pmatrix} - ((1 - eq) \times (bbs + pot))\right)$

Figure 69 – Hermes expected return algorithm.

Table 33 – ***Starting cards classification for Hermes. 1 for top scored hands and 8 for low scored hands. Hands without classification in this table are considered unplayable thus Hermes folds immediately when holding such hands.***

		Offsuit												
		A	**K**	**Q**	**J**	**T**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**
Suited	**A**	1	1	2	2	3	5	5	5	5	5	5	5	5
	K	2	1	2	3	4	6	7	7	7	7	7	7	7
	Q	3	4	1	3	4	5	7						
	J	4	5	5	1	3	4	6	8					
	T	6	6	6	5	2	4	5	7					
	9	8	8	8	7	7	3	4	5	8				
	8				8	8	7	4	5	6	8			
	7							8	5	5	6	8		
	6								8	6	7	7		
	5									8	6	6	7	
	4										8	7	7	8
	3												7	8
	2													7

Finally, the Hermes game playing algorithm is presented in Figure 70. This algorithm uses a rule-based approach that considers the abstracted game state, and the expected return of the current hand, in order to decide either to fold or go all-in. It returns the bet value.

Algorithm $Strategy(Hermes \in N, h \in H)$

$Let\ foldOrCall = 0$
$Let\ allin = s(p(h), h)$

$Let\ opp$ = the last playing opponent that went all-in. If none, select the last playing opponent that raised. If none, select the last playing opponent. If none, select the player in the dealer position.

$Let\ pos$ =the Hermes's position in table. It can be bb (if the Hermes agent is the big-blind), sb (the small-blind position), btn (Hermes is the dealer – last to act), co (cut-off position – before dealer) and utg (under the gun position – first to act).

$Let\ pos_{opp}$ = the opp position in table (with the same possible values as the Hermes's position).
$Let\ er = ExpectedReturn(Hermes, opp, h)$
$Let\ gameState$ =the game's state according to Table I.

if $hclass(P_{hero}) = 1$**then**
  return allin
else if $hclass(P_{hero}) = 2 \wedge er \geq 0$ **then**
  switch *gameState*
    case unopened
      if $pos = sb$ **then**
        return *rand_real_between*(0.0, 1.0)>0.4?allin:fold
      else if $pos = co \vee pos = btn$ **then**
        return allin
      end if
    case limped ∨ allin
      if $pos = bb \vee pos = sb$ **then**
        return allin
      end if
    case limps
      if $pos = bb$ **then**
        return allin
      end if
    case raised
      if $pos = bb \vee pos = sb \vee pos = btn$ **then**
        return allin
      end if
  return foldOrCall
else if $hclass(P_{hero}) = 3 \wedge er \geq 0$ **then**
  switch *gameState*
    case unopened
      if $pos = btn \vee pos = sb$ **then**
        return allin
      end if
    case limped ∨ raised
      if $pos = bb$ **then**
        return allin
      end if
  return foldOrCall
end if

if $pos = bb \wedge gameState = raised \wedge (pos_{opp} = btn \vee pos_{opp} = sb) \wedge fold3bet(opp) \geq 0.5$ **then**
  return allin
else if $pos = bb \wedge gameState = raised \wedge pos_{opp} = btn \wedge fold3bet(opp) \geq 0.5$ **then**
  return allin
end if

return foldOrCall

Figure 70 – Hermes game playing algorithm.

6.5 Summary

This chapter presented the methodologies for Poker game playing with emphasis for the two agents' architecture: Lucifer and Hermes. Both these agents are validated in Chapter 7.

Chapter 7

Validation

This chapter presents the results obtained by the two agent architectures described in Chapter 6. Both agents have very different purposes: Hermes was built to play online and be profitable against humans; Lucifer was an entry to the ACPC competition – a competition that validates the agents in a more theoretical way. For the second case, the agent participated in the multiplayer Kuhn Poker track, a variant of Poker that is so small in terms of search space that enables the developed approaches to better concentrate on the opponent modelling aspects of the game – the lower number of information sets means that less games are needed to determine the best agent in the long run, which reinforces the importance of opponent modelling.

7.1 Online Game Playing (Hermes)

Given that the Hermes agent implementation only plays in a single table at a time and given that it was playing against humans, the result extraction is very time consuming. Even so, the results of 3814 games were extracted[37] (see some statistics in Table 34).

The overall profit of the agent was 1.48 big-blinds (minimum bets) for each 100 games. Since we performed the experiments in tables where the blinds were 0.02€, the agent made an overall absolute profit of 1.13€. Considering that in each game the

[37] Disclaimer: At the time the tests were performed, the use of agents was not illegal in the country where those tests occurred. Moreover, the authors did not find any mention in the software TOS against it. Even so, the account in which the tests were performed was closed short after the tests.

agent had to pay an average 5% commission over the amount of money that was bet, these results can be considered good. Moreover, this particular online casino refunds 20% of the money paid on commissions, at the end of the month, when the player is profitable. This allowed for the agent to make an extra absolute profit value of 7.63€, making a total profit of 8.76€. This results in a final average profit of about 11.5 big-blinds for each 100 games.

Table 34 – Some statistics about the hand played by the Hermes agent.

Feature	Value
Number of hands	3814
VPIP	9.3
PFR	9.0
3Bet	8.9
Winnings	1.13€
Bb/100 games	1.48
Avg. All-in EV	54.6%
Avg. Pre-flop All-in EV	54.3%
Avg. Flop All-in EV	57.0%

7.1.1 All-time results

A graphical representation of the hands played and the agent's profit balance overtime is shown in Figure 71. In this chart we consider that the commission refund function is linear.

As can be observed in Figure 71, the agent's total money balance increases overtime, ending up in a final absolute profit of 8.76€. In this graph, besides the global profit and the commission refunding profit, the showdown and non-showdown profit are also differentiated. The showdown profit includes money lost or won in all games where the agent decided to bet and at least one of the opponents covered that bet. Non-showdown profit includes all money lost when the agent folds or all the money won when the agent goes all-in and all opponents fold.

One important concept to understand for these results' analysis is the definition of stealing and defending blinds. The blinds are mandatory bets that are made by some players before the game begins and before they see their cards. Since that card dealing is a random event, it means that players sometimes are spending money on hands with very low rank. Therefore, defending blinds means to not to waste the

blinds money when the starting hand is good (or convince the opponents that it is good); stealing blinds means to be able to interpret when an opponent that did a blind bet has a weak hand, and therefore doing a high bet to make it forfeit that hand. This is rather important for this type of agent, because it only plays with information from the Pre-Flop, which means that a god amount of the agent's profit comes from blinds.

A conclusion that can be taken from this graph is the importance of stealing and defending blinds (see Section 7.1.4). Since the agent is a tight player (it only raises on a very small number of hands), it ends up folding 0.02€ or 0.01€ too many times, when it is the blinds position. This results in the agent losing too much money (Non-showdown winnings). The only way to reduce these losses would be to play in other rounds instead of Pre-Flop. Being a less tight agent would probably reduce the showdown games earnings.

However, it is possible to observe a slight difference in the non-showdown line, after the 2800 hands, where the gradient starts to decrease. The reason behind this is the gradual improvement of the agent's evaluation on the opponents' pre-flop steal ability. The authors believe that the results will improve if the agent played even more. However, the profit already made by the agent in the showdown winnings compensates its lack of defending blinds ability.

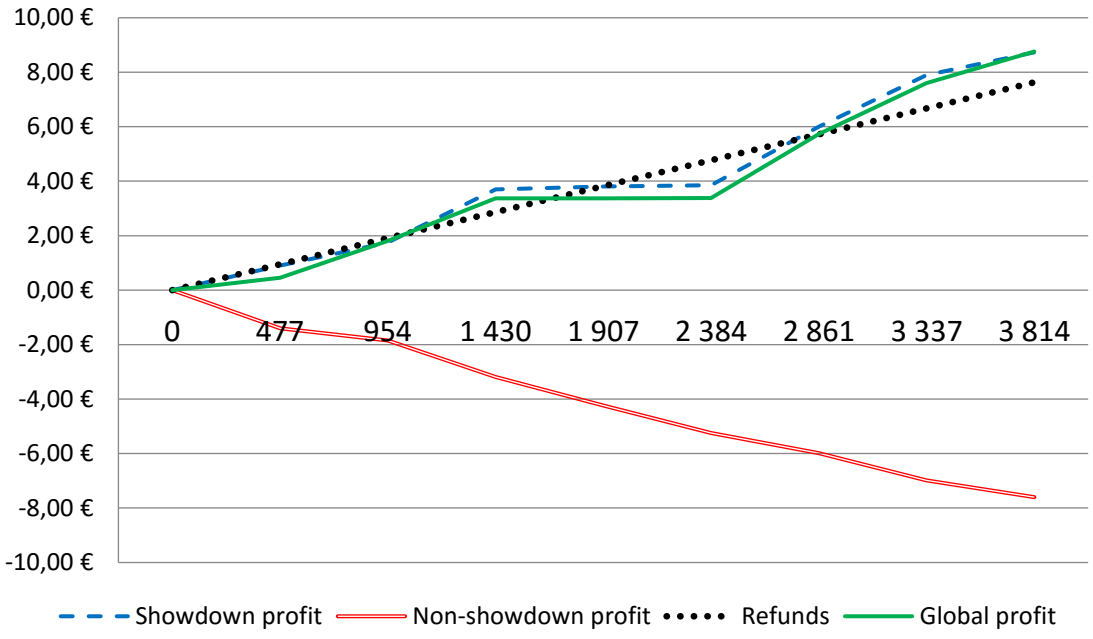


Figure 71 – Hermes's all time profit

7.1.2 Playing style analysis

In order to analyse the agent's playing style, Table 34 presents some relevant statistics that summarize the agent's online performance in this experiment. These statistics do not include the commission refunds, which are dealt independently. Now, each statistic meaning is described:

- **Number of hands** – the total number of Poker games played in this experiment.
- **VPIP (Voluntary Put money In Pot)** – indicates the percentage of games where the agent bets, excluding the money bet when the agent was in the blinds positions. As expected and as said earlier, since the agent's strategy is tight, the agent only went all-in in about 9% of the games.
- **PFR (Pre-flop raise)** – number of times the agent raises any amount in the Pre-Flop round. Since the agent's strategy only considers the Pre-Flop round, this value is very similar to VPIP. The agent only plays after the Pre-Flop if it can get a free Flop, which means that the agent is in the big-blind position and none of the opponents bet any amount, thus enabling the agent to just call the hand.
- **3Bet** – the number of times the agent raises after any opponent has raised. As expected, this measure is also similar to **VPIP** since the agent usually only plays in table positions where it decides the action after other players.
- **Winnings** – the absolute winnings excluding the commission refunds. These winnings, depending on the value of the blinds, are the ones that indicate if the agent is entitled to commission refunding.
- **Bb/100 games** – the number of big-blinds (minimum bets) won for each 100 games. This is the common measure that is used to evaluate if a player is good or not. The way the value of this measure has to be looked depends greatly on the value of the blinds. For instance, for games with blinds of 0.50-1.00€, a good player should have about 7Bb/game. In 0.02-0.01€ games, a good player should have about 10Bb/game [82].
- **Avg. All-in EV** – the expected value when the agent goes all-in. This measure is relative to the investment made by the agent. In these experiments, the

average EV is 54.6%, which means that when the agent goes all-in, it has a positive profit of 54.6% of the amount that was bet. For this stat, we indicate its average value in the game, in the Pre-Flop and after the Pre-Flop.

From these stats it must be highlighted the positive expected value for all-in actions in all rounds. This means that when the agent goes all-in, it profits in average more than 50% of its investment.

7.1.3 Playing with table position

Now let's analyse the agent's ability to play in different positions in the table (Table 35). Again the profit made from refunds is not being considered. The events on each position in this particular experiment are:

- Small-blind – the player has to pay 0.01€ at the start of the game without seeing its cards. It is the penultimate player to choose his/her action.
- Big-blind – the player has to pay 0.02€ at the start of the game without seeing its cards. It is the last player to act.
- Early – no blinds. It is one of the first players to act. This position is disadvantageous because the player has to act without any feedback from his/her opponents.
- Button – also known as dealer position. In the Pre-Flop is antepenultimate player to act or the last if only two players are playing. It is the most advantageous position since the player does not have mandatory bets and he/she can get feedback from the actions of most of the opponents.
- Cutoff – position just before the button.
- Middle – positions between the last early and the cut-off.

The conclusions that we can take from these results are that playing in positions where blind bets are made, will always pose the threat of losing money (especially when the blinds are so low); the only way to lessen this leak is to improve the evaluation on the stealing probability. The agent's performance in each position is overall satisfying, showing a profit on almost all positions excluding the blinds and the

cut-off. The cut-off negative income is probably due to the results' variance (low number of games), since the expected value in that position is positive. A very satisfying statistic to highlight is the average all-in percentage which is above 50% in all positions. This surely proves that the more hands the agent plays the more profit it will attain. The small blind VPIP is the highest among all, which means that the agent tries to steal the big blind every chance he sees fit. Also the highest average all-in percentage comes from the early position, which is expected since it is the position the agent plays more seldom, making its hand ranges a lot stronger. It is also possible to observe the following facts:

- The average EV for all positions where the agent has to bet blinds is negative, as expected (because the agent was to put money even with hands that it will forfeit; the only way of playing with those hands is if it is a free call). However, the global average EV is positive, which means that for the sum of all positions the agent is profitable.
- The more similar the actual Profit is to the EV, the more stabilized are the statistics about the agent's game play. This means that, in this case, the profit was over than what was statistically expected, because the agent was *"lucky"* or because the opponents fold their hand in response to more aggressive moves by the agent.

Table 35 – Hermes's playing style statistics

Position	Hands	Profit	EV	VPIP%	PFR%	3Bet%	Avg All-In
Small blind	695	-1.43€	-2.80€	14.0%	13.5%	11.5%	51.8%
Big blind	701	-4.47€	-4.73€	10.4%	10.1%	9.5%	52.4%
Early	411	1.54€	1.86€	5.6%	5.6%	-	64.1%
Middle	620	0.77€	1.34€	6.8%	6.8%	6.5%	57.5%
Cut-off	685	-0.34€	1.18€	7.2%	6.9%	5.5%	56.0%
Button	702	5.06€	3.17€	10.0%	9.5%	6.9%	55.6%
Totals	3814	1.13€	0.02€	9.3%	9.0%	8.9%	54.6%

7.1.4 Stealing and defending blinds

In Table 36 the agent's results when defending and stealing blinds situations (without profit refunds) are demonstrated.

Table 36 – Hermes's defending and stealing blinds statistics

Type	Hands	Profit	EV	Fail | All-In EV%
Stealing	102	3.59€	1.81€	55.1%
Defending	51	0.51€	1.60€	49.3%

Stealing blinds is a situation where the agent raises at Cut-off, Button or small-blind positions. Stealing blind results are extremely positive, since the agent's objective is to steal blinds while taking into account the fold chance of the opponents, since it does not play in other rounds. When the steal attempt fails, the most likely reason for that is the agent not accurately knowing yet the opponent's range. However, the agent has still a very high relative expected value (55.1%) when it fails to steal the blinds and goes all-in. Giving the small amount of the blinds (used in these tests), this probably means that there is still a good margin for stealing more blinds by bluffing more, because for the presented expected values, it means that the agent only played premium hands (which means that it probably folded too much). These results show the high importance the steal factor has in the Poker game (3.59€ in only 2.67% of the games has a huge significance). In Figure 72 it is possible to observe the positive growing rate (about 3.5%) of the profit in these situations.

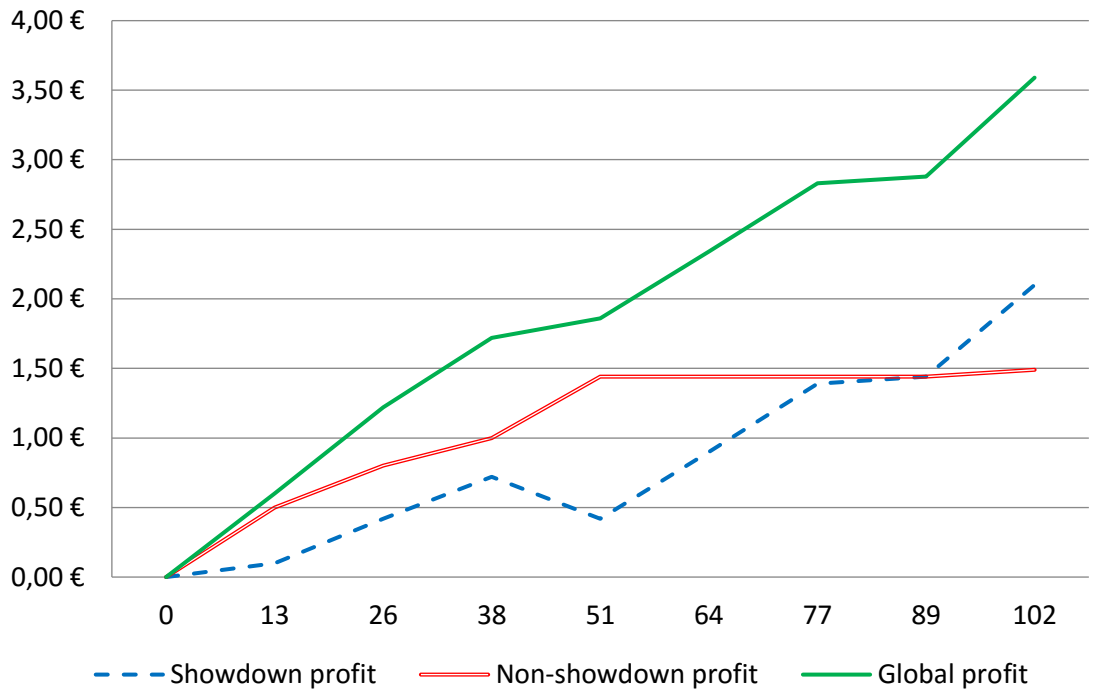


Figure 72 – Hermes's stealing blinds results

Defending a blind is a situation where the agent is in a table position where it has to bet blinds, and has to reply to a raise from another player. In Table 36 the results when the agent tries to defend the blind by going all-in are demonstrated. Here it is possible to see that the calculations for expected value were fairly accurate, since among all the all-ins made, the average all-in expected value percentage is 49.3%, meaning when the agent is called it will still have a good winning rate, and when it does not get called it wins the blinds plus the raises of the opponents. The expected value from these plays is higher than the actual winnings: this mean that the agent played well, despite of the variance not being on his side. Nevertheless it is still a small amount of hands, and in the long run the winnings would possibly even with the expected value. It is a very good expected value of 1.60€, since the agent bets 0.20€ at a time.

It is possible to conclude by these results that when the agent defends blinds, it defends them correctly. However, by looking again at the results in Figure 71, it is possible to assert that the agent either just does not defend the blinds enough times or that no more profit can be made from this choice.

7.1.5 Results against particular players

In Table 37, the results of Hermes against the players that allowed it to make more profit are presented. The most significant players to note here are the ones which have a number of hands higher than 100, namely: Player2, Player9 and Player10. These three players show fairly good statistics, making them tight aggressive players (VPIP < 28%; most winning players are tight aggressive [16]), and still the agent was able to exploit them and make a good positive profit over time.

In Table 38 the results of the players who gave negative profit to the agent are shown. Looking at the top five most unprofitable opponents, their stats vary from a very tight aggressive player, Player15, to a very loose aggressive player, Player13. A quick look at the hands played against these players allows us to verify that some of these opponents (Player15 or Player20) have dominated the agent's strategy. Others, like Player11 or Player16 may be justified to the results high variance.

Table 37 – Hermes's against the 10 most profitable opponents

Opponent	Hands	VPIP	PFR	Profit
Player1	14	42.9%	28.6	1.49€
Player2	271	17.7%	13.7%	1.00€
Player3	14	64.3%	21.4%	0.97€
Player4	41	82.9%	9.8%	0.79€
Player5	39	41.0%	12.8%	0.76€
Player6	5	40.0%	0.0%	0.73€
Player7	16	31.3%	18.8%	0.69€
Player8	45	57.8%	0.0%	0.62€
Player9	455	24.2%	11.2%	0.60€
Player10	860	19.8%	16.3%	0.56€

Table 38 – Hermes's against the 10 less profitable opponents

Opponent	Hands	VPIP	PFR	Profit
Player11	54	55.6%	25.9	-1.07€
Player12	180	31.1%	12.2%	-0.86€
Player13	38	84.2%	23.7%	-0.62€
Player14	148	26.4%	23.0%	-0.60€
Player15	277	27.8%	19.9%	-0.59€
Player16	67	22.4%	20.9%	-0.59€
Player17	136	20.6%	16.2%	-0.56€
Player18	224	20.5%	19.2%	-0.56€
Player19	25	72.0	40.0	-0.46€
Player20	774	15.0%	13.0%	-0.46€

7.1.6 Summary

As stated before, the Hermes implementation required background knowledge and expertise of a domain expert on the Texas Hold'em variant of Poker. Nevertheless, despite the strategy not being (yet) as good (profitable) as the one from the original player, the authors believe a great step was done towards the goal of making Poker agents more profitable than the best human players, by showing that it is now possible to create a winning agent. The most surprising aspect was the agent surpassing most of the human players found online, just by considering the Pre-Flop stage of the game. Some suggestions for possible improvements would be working on the blind stealing ability on the three positions fit to do so: big blind, small blind and button. The agent can also be improved in the matter of autonomy at the tables, for instance, leaving a table when holding more than 20 big blinds (the used Poker Bot software does not

support this – see Section 4.3), entering a new table where the minimum players is 4, leaving a table when it falls below 4 players, by this way optimizing the short-stack strategy (which is proved to work better in tables with 4 to 6 players). In future work the agent should also be tested in games with higher stakes, since they usually present more skilled players. Another important feature to add is the ability to play in simultaneous tables to allow for the agent to get profit much faster.

7.2 AAAI 2014 Competition (Lucifer)

7.2.1 Competition rules and goal

The Lucifer agent participated in the Kuhn 3P track of the Annual Computer Poker Competitions that was held in 2014. The Kuhn track was held for the very first time in 2014 with the objective of encouraging teams to invest time in opponent modelling techniques. The Kuhn Poker game is perfect to validate results in opponent modelling because it is a very short game with a very small number of possible information sets. The existence of 3 players also prevents the participants of using exclusively Nash-Equilibrium based solutions, as current techniques do not give mathematical guarantees that the agents are in fact in an equilibrium stage. The rules of the competition[38] were:

- **Game**: Limit Kuhn Poker. There is a single round of betting in Kuhn poker. Each player first antes a single chip and is dealt a card from a deck containing one jack, queen, king and ace. The first player then has the option to check, or bet an additional chip. When facing a bet, a player can call the bet or fold. That is, only a single bet is allowed by any player. At showdown, the highest card wins the entire pot. The ace is the highest card.

- **Competition Format**: Series of 3-player duplicate matches. Introduced in the 2009 competition, multiplayer duplicate generalizes the heads-up duplicate format for the 3 player matches. If we consider that there are 3 possible seats that each bot can sit in, and 2 different relative orderings of the other 2 bots given the position of one bot, then there are six total possible configurations of

[38] From: http://www.computerpokercompetition.org/index.php/competitions/rules/96-2014-rules

3 players at a given table. If we choose to play N hands per match then the following system will assure all players rotate through all possible seats and relative orderings:

- Seat the players in some ordering, say bot 1 is the small blind, bot 2 the big blind and bot 3 the button
- Play N/6 hands using standard poker rules: after every hand the button and blinds rotate one seat to the left
- Reset the memory of the bots
- Rotate the seating of the players to the left, so in our example bot 1 is now on the button, bot 2 is SB, bot 3 is BB
- Play N/6 hands again, dealing the same cards as before to the same seats as before (bot 1's first hand is now bot 3's first hand from round 1)
- Reset the bots again
- Rotate once more
- Play the same N/6 hands again
- Reseat the players in the other relative ordering - bot 1 SB, bot 3 BB, bot 2 button
- Repeat the above process of dealing out the same N/6 hands to the same seats, resetting the memories and rotating the bots between rounds

— **Hand Per Match:** 3000

— **Stack Sizes:** Infinite

— **Bet Size:** 1 chip

— **Ante Sze:** 1 chip

— **Showdown Mucking:** No

— **Illegal Actions:** Any illegal action is interpreted as a check/call.

— **Winner Determination:** total bankroll. The total bankroll winner determination rule encourages competitors to submit agents that can do one thing: maximize their total winnings across all opponents.

7.2.2 Competition results

The Lucifer agent performed very well in the 2014 competition, getting the 2nd place in the competition, only losing to the Alberta's Computer Poker Research Group bot (see Figure 73). The Hyperborean used a CFR based approach and HITSZ used case based reasoning approach.

Hyperborean[39] (University of Alberta, Canada)

Lucifer (LIACC, University of Porto, Portugal)

HITSZ (School of Computer Science and Technology HIT, China)

Figure 73 – AAAI Computer Poker Competition 2014 – highest ranked teams in the Kuhn 3P track.

The full results of the competition are presented: in Table 39 – it demonstrates the global results of the competitions by giving all combinations of three agents and providing the global bankroll and variance for matches between those agents; Table 40 – a different view for Table 39, which displays the results per match; Table 41 – it demonstrates the completion global results when combining the teams that participated with different agents for each position in the table (in this case only the Hyperborean agent played with 3 different programs). As it can be observed from the tables, the winner of this competition was the Hyperborean agent, because it never had prejudice in any match. All tables express their results on average absolute gains (in blinds per 1.000 games) and their respective variances.

[39] With 3 entries, one per different match

Table 39 – Results from the top scored teams in the 2014 three player Kuhn poker AAAI competition[40].

Opponents		Competitors[41]					
		HB.RMPUE	**HB.BFO**	**HB.AEWRM**	**LUCIFER**	**HITSZ_CS**	**KYH[42]**
Lucifer	HB.RMPUE					-17.40 ± 3.05	-74.22 ± 6.95
HITSZ_CS	HB.RMPUE				**-21.33 ± 4.71**		-61.98 ± 5.92
KYH	HB.RMPUE				**24.44 ± 5.93**	-12.78 ± 5.54	
Lucifer	HB.BFO					-6.52 ± 2.73	-78.18 ± 10.31
HITSZ_CS	HB.BFO				**-3.12 ± 3.79**		-35.50 ± 9.78
KYH	HB.BFO				**50.42 ± 7.15**	-16.74 ± 5.96	
Lucifer	HB.AEWRM					-8.80 ± 3.49	-46.50 ± 6.87
HITSZ_CS	HB.AEWRM				**-7.15 ± 5.24**		-44.95 ± 9.11
KYH	HB.AEWRM				**20.78 ± 5.99**	18.35 ± 2.89	
HITSZ_CS	Lucifer	38.73 ± 4.27	9.64 ± 4.96	15.96 ± 4.05			-17.25 ± 7.04
KYH	Lucifer	49.78 ± 5.23	27.76 ± 10.21	25.72 ± 7.90		-6.47 ± 2.26	
KYH	HITSZ_CS	74.76 ± 3.58	52.24 ± 9.74	26.60 ± 7.16	**23.72 ± 5.23**		
Average		54.4231	29.8803	22.7593	15.0223	-7.0328	-43.6771

Table 40 – Played matches results in the 2014 three player Kuhn poker AAAI competition.

Match ID	HB.RMPUE	HB.BFO	HB.AEWRM	LUCIFER	HITSZ_CS	KYH
M01	38.73 ± 4.27			**-21.33 ± 4.71**	-17.40 ± 3.05	
M02	49.78 ± 5.23			**24.44 ± 5.93**		-74.22 ± 6.95
M03	74.76 ± 3.58				-12.78 ± 5.54	-61.98 ± 5.92
M04		9.64 ± 4.96		**-3.12 ± 3.79**	-6.52 ± 2.73	
M05		27.76 ± 10.21		**50.42 ± 7.15**		-78.18 ± 10.31
M06		52.24 ± 9.74			-16.74 ± 5.96	-35.50 ± 9.78
M07			15.96 ± 4.05	**-7.15 ± 5.24**	-8.80 ± 3.49	
M08			25.72 ± 7.90	**20.78 ± 5.99**		-46.50 ± 6.87
M09			26.60 ± 7.16		18.35 ± 2.89	-44.95 ± 9.11
M10				**23.72 ± 5.23**	-6.47 ± 2.26	-17.25 ± 7.04

[40] Adapted from http://www.computerpokercompetition.org/index.php/competitions/results/105-2014-results?showall=&start=4. Results are expressed in number of big-blinds for each 1000 games.

[41] HB.RMPUE (hyperborean3pk.RMPUE), HB.BFO (hyperborean3pk.BFO) and HB.AEWRM (hyperborean3pk.AEWRM) are all agents from the same competitor. The competition rules allow participants to use different programs in a different table seat.

[42] Full name: KuhnYouHandleIt

Table 41 – Global compressed results by team from the top scored teams in the 2014 three player Kuhn poker AAAI competition.

Match ID	Hyperborean	LUCIFER	HITSZ_CS	KYH
C01	21.44 ± 4,43	**-10.53 ± 4,58**	-10.91 ± 3,09	
C02	34.42 ± 7,78	**31.88 ± 6,36**		-66.30 ± 8,04
C03	51.20 ± 6,83		-3.72 ± 4,80	-47.48 ± 8,27
C04		**23.72 ± 5,23**	-6.47 ± 2,26	-17.25 ± 7,04
Average	35.69 ± 6,34	**15.02 ± 5,39**	-7.03 ± 3,38	-43.68 ± 7,78

The results of Lucifer agent were overall good. Lucifer only had negative prejudice when one of the Hyperborean bots was participating in the game and when the other one was HITSZ_CS (see Table 40 in matches M01, M04 and M07), because Lucifer could exploit HITSZ_CS (see M10) but not as well as the KYH, which means that most of the money lost goes to Hyperborean. However, Lucifer was able to make a very good profit in all other games: it even surpassed Hyperborean's profit once in M05. When Hyperborean was not participating (see match C04 in Table 41), Lucifer's victory was unquestionable (none of the other competitors was able to make any profit). In other matches, where HITSZ_CS was not participating, Lucifer even had a game where its results were better than the Hyperborean (M05), which means that Lucifer made a better model of the KYH agent than HITSZ_CS.

By combining the completion overall results in Table 41 (because Hyperborean had 3 agents, one of each game position), it is possible to see the Lucifer had an unquestionable 2nd place because its final bankroll is very far away from the 3rd place (which had negative profit). The overall Lucifer's profit in this competition was 15 blinds for each 1000 games, which means that it is capable of increasing its bankroll by 1.5% in each game. Considering the simplicity of the played variant, this result is very good, since luck has a huge impact on this game (it is highly likely that one of the players will get an Ace, making it the virtual winner of game) – this game is won by who folds better.

7.3 Summary

This chapter summarized the results obtained by the two developed agent architectures during this research work. The obtained results are very promising, especially the results on online matches where, for the first time reported, it is shown that an agent can be profitable in multiplayer tables against human players. Further testing should be made in the future, against more competitive human players (the limitation here are the amounts of the bet at tables with higher blinds, which would require funding for these tests to be done). As for the ACPC competition, the results demonstrate that the opponent modelling is the key for being successful at multiplayer tables.

Chapter 8

Conclusions

Although there is a lot of finished research on Computer Poker there is still no known Poker agent capable of beating the best human players in the Texas Hold'em variant in full multiplayer tables (especially the No-Limit variant). Recent approaches like the Cepheus agent show that such goal is not impossible and that we are getting closer. However, there are still many challenges, most of which related to the size of the problems and the current limited capacity of our hardware to deal with such huge amounts of data. This might be easier to address one day with the popularization of quantum computing (which will enable us to model problems in a different but more capable way to deal with huge amounts of data) or the simulation or realization of human like traits in computers, such as intuition, a skill humans use in so many problems (like Poker) without us being able to explain how.

This research embraced many different areas, in order to help the development of the Computer Poker research domain. The authors believe that the contributions can be divided into two different parts: supporting tools (Chapter 4) and domain advancements (Chapter 5, Chapter 6 and Chapter 7). The first part contributed to the expansion of the Computer Poker research domain by analysing and describing several tools that not only allow for a faster progression in this domain but also introduce new challenges and goals to be achieved. Moreover, it is innovative because the simulation and modelling area in Poker had very little research until now. The second part

consists of improvements to already existing approaches that can be used to further enhance current methodologies.

8.1 Contributions

The contributions that are considered to be the key contributions of this thesis are:

- A new **simulation system** that supports computer Poker research by accommodating researchers needs. This system includes a new **language for specifying custom Poker variants**, and a very simple **general Poker game playing agent**.
- An **online poker playing bot** software and API which allows Poker playing agents to compete against human players in real money games. This is done without the knowledge of human players so as to eliminate the possible psychological effects.
- New approach, with minimum memory usage, that greatly **speeds-up the computation of hand strengths**, called **Average Rank Strength**. The same method can be used to compute other prediction measures in Poker.
- A new configurable and **domain-independent abstraction algorithm (RGU)** based on the average utility of a Nash-Equilibrium profile strategy. The size of the abstraction depends on the computer resources and is completely customizable.
- An algorithm for the **inference of high-level strategies** described in the language [45]. It was demonstrated that this methodology works by empirically inferring several strategies from human game playing data.
- A new live **opponent modelling** methodology named **K-Current-Best-Utility** strategy, which allows an agent to dynamically adapt to the current opponents' strategies, almost without any storage, which was validated empirically in the AAAI Computer Poker Competition.
- Some **optimizations in the Counterfactual Regret Minimization** algorithm, namely a non-recursive implementation (which increases the amount of

necessary memory but greatly reduces the computation time) and decision tree pruning optimizations which can greatly reduce the computation time with almost no impact in the generated strategies.

— An agent architecture that got the **first reported results of an agent being profitable** in online games against several human players. The reported matches were played with real money.

8.2 Goals Achievement

Regarding the completion of this thesis's goals, we now summarize the goals accomplishment level. More or less, all goals have been addressed and have some degree of achievement.

— ***Goal**: Explore how methodologies used on the Computer Poker domain can potentially be used or at least hint to the solution of other AI-related problems*.

The final version of this thesis focused more on Computer Poker itself than its applications to other knowledge areas. As explained in Sections 2.1 and 2.5, Poker itself is import enough (in terms of public interest) to be the main focus of this thesis but, however, the authors feel that there is room to improve on these aspects. Nevertheless, the developed improvements on the **Counterfactual Regret Minimization** algorithm (an algorithm that is domain independent), the **K-Current-Best-Utility** method and the **RGU** abstraction represent contributions that are domain independent and can be adapted to (at least) a great deal of sequential games.

— ***Goal**: Create domain validation methodologies and tools for better assessment of scientific advances.*

As demonstrated in Chapter 4, the developed tools allow for the software agents to be tested and validated in almost all important aspects of the Computer Poker domain. The main limitation of the developed tools is in the Poker Bot software not supporting the automatic selection of the table, which would make it completely autonomous.

— ***Goal**: Present necessary engineering aspects for the construction of Poker agents as opposed to a more theoretical approach*

The completion of this goal is demonstrated by the implementation of the *Poker Bot Software* described in Section 4.3 and the Lucifer and Hermes agent architectures, respectively on Sections 6.3 and 6.4.

— ***Goal**: Improve the efficiency of current techniques in order to reduce the huge amount of resources that they need*

Two of this thesis contributions helped achieving this goal: the execution time of main state of art algorithm – Counterfactual Regret Minimization – can be greatly reduced through a parallel and linear implementation with a memory usage increase; the ***Average Rank Strength*** can reduce by a 1.000 times the computation time, with an additional small memory usage increase. As described on Section 6.2.2, it is still possible to improve even further the linear CFR execution time and decrease its memory usage, but the hardware that was available to do the tests limited the experiments.

— ***Goal**: Find out how to combine current techniques and technologies to create a Poker agent that finally surpasses human players by being profitable in online multiplayer matches*

This goal was achieved through the implementation of the Hermes software agent, which reported the first positive online results, with fair testing against human players.

— ***Goal**: Overcome the limitations of current methodologies on multiplayer games.*

Until now, there was very little research on multiplayer Poker and the importance of opponent modelling techniques in those type of games. Lucifer agent achieved this goal by introducing the **K-Current-Best-Utility** method and by obtaining good results in the 2014 ACPC competition (see Section 7.2.2).

Regarding the research questions of this thesis, we can outline the answers for them:

— ***Question**: Is it possible to improve current simulation tools for Poker games? If so, will this improvement help on the construction of more competitive Poker playing agents?*

Yes, by deeply analysing the game in all its aspects especially by using expert knowledge and not only the more scientific and theoretical aspects. This thesis

demonstrated the importance of having agents interacting with humans, because more theoretical approaches such as Nash-Equilibrium strategies are still unfeasible for online game play.

— ***Question**: With currently available technology is it already possible for a Poker playing software agent to be profitable in online multiplayer matches with real money bets? If not, what needs to be improved in software agents to do so?*

Yes, now software agents can be profitable online multiplayer matches with real money, against weak human players. However, it is still unclear if those agents can beat highly skilled players.

— ***Question**: In which way can abstraction techniques be improved in order to be domain-free and to better represent their corresponding unabstracted games?*

Abstraction techniques for sequential games should focus on the utility obtained from possible plays. On possible domain free implementation resides on using the utility of similar but smaller games that are tractable with current hardware.

— ***Question**: How is it possible to reduce the large number of resources needed by current techniques without compromising the final results?*

It is possible by using as less recursivity as possible on the methodologies. On this domain the recursive algorithms are easier to explain, implement and represent. However, they lack capacity of being easily applicable without a huge amount of computational resources.

8.3 Future Work

There are several improvement points that can be done in this work. Some potential improvements are:

- Try to apply the **K-Current-Best-Utility** in online games with different agent architectures, by using teams of computer programs as was done in previous works such as [30].
- Improve the **Average Rank Strength** method in order to make it more generic. This would allow it to be adapted to other Poker variants and be therefore better integrated in the **PGDL System**.

- Some improvements in the tools that were created: enable the Poker bot to **select the playing table** instead of selecting it manually to truly automate the bot software. Allowing **multiple tables at the same time** would also enable software agents like **Hermes** to earn money faster.
- Enable the **Hermes** agent to play in Post-Flop rounds of the game of No-Limit Texas Hold'em – this could make the agent much more profitable.
- Improvements in **Linear CFR** by using a better way to store very large sparse arrays.

References

[1] R. Kurzweil, *"The Singularity is Near"*, Publisher: Gerald Duckworth & Co Ltd, 2006 (Book)

[2] G. E. Moore, *"Cramming more components onto integrated circuits"*, Proceedings of IEEE, vol. 86, no. 1, pp. 82–85, 1998

[3] F. Schürmann et al, *"The Blue Brain Project: building the neocortical column"*, Proceedings of CNS 2007, vol. 8, no. 2, pp. 109, 2007

[4] H. Berliner, *"Kasparov Versus Deep Blue: Computer Chess Comes of Age"*, Publisher: Springer-Verlag New York, 1997 (Book)

[5] J. Doughney and T. Kelleher, *"The impact of poker machine gambling on low-income municipalities A Critical Survey of Key Issues"*, Victoria University of Technology, 1999 (Technical report, unpublished)

[6] J. A. McKenna, *"Beyond Tells: Power Poker Psychology"*, Publisher: Stuart (Lyle) Inc., 2005 (Book)

[7] A. Schoonmaker, *"The psychology of poker"*, Publisher: Two Plus Two, 2000 (Book)

[8] D. Billings, *"Computer Poker"*, Master Thesis, University of Alberta, 1995

[9] J. Nash, *"Equilibrium points in n-person games"*, Proceedings of the National Academy of Sciences of the United States of America, vol. 36, no. 1, pp. 48-49, 1950

[10] M. Johanson, *"Measuring the Size of Large No-Limit Poker Games"*, University of Alberta, 2013 (Technical report, unpublished)

[11] E. Oliveira et al, *"Emotional advantage for adaptability and autonomy"*, Proceedings of AAMAS '03, pp. 305, 2003

[12] F. Huang, X. Luo, H.-F. Leung, and Q. Zhong, *"Games Played by Networked Players"*, Proceedings of Intelligent Agent Technologies (IAT), 2013, vol. 2, pp. 1–8.

[13] R. Myerson, *"Game theory: analysis of conflict"*, Publisher: Harvard University Press, 1991 (Book)

[14] J. Neumann, *"Zur theorie der gesellschaftsspiele"*, Publisher: Princeton University Press, 1944 (Book)

[15] J. Smith et al, *"The Logic of Animal Conflict"*, Nature 246, 15, 1973

[16] D. Sklansky, *"The Theory of Poker: A Professional Poker Player Teaches You How to Think Like One"*, 4th Edition, Publisher: Two Plus Two, 2007 (Book)

[17] D. Billings et al, *"The challenge of poker"*, Artificial Intelligence, vol. 134, no. 1–2, pp. 201–240, 2002

[18] L. F. Teófilo, *"Building a poker playing agent based on game logs using supervised learning"*, Master Thesis, University of Porto, 2010

[19] M. Johanson et al, *"Evaluating state-space abstractions in extensive-form games"*, Proceedings of AAMAS '13, pp. 271–278, 2013

[20] A. Gilpin et al, *"Better automated abstraction techniques for imperfect information games, with application to Texas Hold'em poker"*, Proceedings of AAMAS 07, pp. 1, 2007

[21] M. Johanson et al, *"Finding Optimal Abstract Strategies in Extensive-Form Games",* Proceedings of AAAI-12, pp. 1371–1379, 2012

[22] L. F. Teófilo et al, *"A Simulation System to Support Computer Poker Research"*, Proceedings of MABS'12 - 13th International Workshop on Multi-Agent Based Simulation at AAMAS, pp. 81–92, 2012

[23] M. Zinkevich, M. Bowling, and N. Burch, *"A new algorithm for generating equilibria in massive zero-sum games"*, Proceedings of AAAI, pp. 788–793, 2007

[24] M. Bowling, N. Burch, M. Johanson, and O. Tammelin, *"Heads-up limit hold'em poker is solved"*, Science Journal, vol. 347, no. 6218, pp. 145-149, 2015

[25] D. Billings, *"Algorithms and assessment in computer poker"*, Ph.D. Thesis, University of Alberta, 2006

[26] R. Gibson et al, *"Regret Minimization in Games and the Development of Champion Multiplayer Computer Poker-Playing Agents"*, Ph.D. Thesis, University of Alberta, 2014

[27] J. Rubin, *"On the Construction, Maintenance and Analysis of Case-Based Strategies in Computer Poker",* Ph.D. Thesis, University of Auckland, 2013

[28] M. Zinkevich et al, *"The 2006 AAAI Computer Poker Competition"*, Journal International Computer Games Association, no. 29, pp. 166–167, 2006

[29] J. Rubin et al, *"Computer poker: A review"*, Artificial Intelligence Journal, vol. 175, no. 5–6, pp. 958–987, 2011

[30] L. F. Teófilo et al, *"Building a No Limit Texas Hold'em Poker Playing Agent based on Game Logs using Supervised Learning"*, Proceedings of 2nd International Conference on Autonomous and Intelligent Systems, pp. 73–83, 2011

[31] D. Billings et al, *"Using probabilistic knowledge and simulation to play poker"*, Proceedings of AAAI/IAAI, pp. 697-703, 1999

[32] B. Sheppard, "World-championship-caliber Scrabble", Artificial Intelligence Journal, vol. 134, no. 1, pp. 241-275, 2002

[33] A. Van der Kleij, *"Monte Carlo Tree Search and Opponent Modeling through Player Clustering in no-limit Texas Hold'em Poker"*, Master Thesis, University Groningen, Netherlands, 2010

[34] G. Broeck et al, *"Monte-Carlo Tree Search in Poker Using Expected Reward Distributions"*, Proceedings of ACML '09, pp. 367–381, 2009

[35] M. Ponsen et al, *"Integrating Opponent Models with Monte-Carlo Tree Search in Poker"*, Proceedings of Interactive Decision Theory and Game Theory at AAAI, 2010

[36] M. Johanson, *"Robust Strategies and Counter-Strategies: Building a Champion Level Computer Poker Player"*, Master Thesis, University of Alberta, 2007

[37] F. Oliehoek, *"Game theory and AI: a unified approach to poker games"*, Master Thesis, University of Amsterdam, 2005

[38] R. J. Vanderbei, *"Linear Programming: Foundations and Extensions (International Series in Operations Research & Management Science)"*, Publisher: Springer, 4th edition, 2013 (Book)

[39] L. F. Teófilo et al, *"Adapting Strategies to Opponent Models in Incomplete Information Games: A Reinforcement Learning Approach for Poker"*, in Autonomous and Intelligent Systems - Third International Conference (AIS2012), pp. 220–227, 2012

[40] M. Johanson et al, *"Data biased robust counter strategies"*, Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics (AISTATS), pp. 264–271, 2009

[41] M. Zinkevich et al, *"Regret Minimization in Games with Incomplete Information"*, Proceedings of Advances in Neural Information Processing Systems 20 (NIPS), pp. 1729–1736, 2008

[42] N. Risk et al, *"Using counterfactual regret minimization to create competitive multiplayer poker agents",* Proceedings of the Ninth International Conference on Autonomous Agents and Multiagent Systems (AAMAS), 2010

[43] M. Lanctot et al, *"Monte Carlo sampling for regret minimization in extensive games",* Proceedings of Advances in Neural Information Processing Systems 22, pp. 1078–1086, 2009

[44] M. Johanson et al, *"Efficient Nash equilibrium approximation through Monte Carlo counterfactual regret minimization"*, Proceedings of AAMAS '12, vol. 2, 2012

[45] L. Reis et al, *"High-Level Language to Build Poker Agents"*, Advances in Intelligent Systems and Computing, vol. 206, pp. 643–654, 2013

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

[47] D. Billings et al, *"Game-tree search with adaptation in stochastic imperfect-information games"*, Computers and Games Volume 3846 of the series Lecture Notes in Computer Science, pp. 21–34, 2006

[48] L. F. Teófilo et al, *"HoldemML: A framework to generate No Limit Hold'em Poker agents from human player strategies"*, 6th Iberian Conference on Information Systems and Technologies (CISTI 2011), pp. 755–760, 2011

[49] A. S. VICTOR, *"Learning in Simplified Poker by clustering opponents"*, Master Thesis, University of Manchester, 2008

[50] A. Sandven et al, *"A case-based learner for poker"*, Proceedings of the Ninth Scandinavian Conference on Artificial Intelligence (SCAI 2006), 2006.

[51] J. Rubin et al, *"Investigating the Effectiveness of Applying Case-Based Reasoning to the Game of Texas Hold'em"*, Proceedings of FLAIRS, 2007

[52] J. Rubin et al, *"Similarity-based retrieval and solution re-use policies in the game of Texas Hold'em"*, Volume 6176 of the series Lecture Notes in Computer Science pp 465-479, 2010

[53] D. Papp, *"Dealing with imperfect information in poker"*, Master Thesis, University of Alberta, 1999

[54] D. Billings et al, *"Approximating game-theoretic optimal strategies for full-scale poker"*, Proceedings of the 2003 International Joint Conference on Artificial Intelligence (IJCAI), 2003.

[55] T. Schauenberg, *"Opponent modelling and search in poker"*, Master Thesis, University Alberta, 2006.

[56] D. Schnizlein et al, *"State translation in no-limit poker"*, Master Thesis, University of Alberta, 2009

[57] *"Pokersource Poker-Eval"*, Available at http://pokersource.sourceforge.net/ (Online)

[58] C. Kev, *"Cactus Kev's Poker Hand Evaluator"*, Available at http://www.suffecool.net/poker/evaluator.html (Online)

[59] E. A. Fox et al, *"Practical minimal perfect hash functions for large databases"*, Communications of ACM, vol. 35, no. 1, pp. 105–121, 1992

[60] Senzee5, *"Paul senzee on software, game development, technology and life", Available at* http://www.paulsenzee.com/2006/06/some-perfect-hash.html (Online)

[61] *"Coding the Whell: Poker Hand Evaluator Roundup"*, Available at http://www.codingthewheel.com/archives/poker-hand-evaluator-roundup (Online)

[62] J. Varho, *"7 Card Poker Hand Evaluation"*, Available at http://jan.varho.org/?p=99

[63] D. Félix et al, *"An Experimental Approach to Online Opponent Modeling in Texas Hold'em Poker*", Proceedings of SBIA '08 pp. 83 – 92, 2008

[64] B. Chen et al, *"The Mathematics of Poker"*, 1st edition, Publisher: Conjelco, 2006 (Book)

[65] D. Sklansky et al, *"Hold'Em Poker for Advanced Players"*, 3rd edition, Publisher: Two Plus Two, 2001 (Book)

[66] D. Brunson, *"Doyle Brunson's Super System: A Course in Power Poker"*, 3rd edition, Publisher: Cardoza Publishing, 2003 (Book)

[67] M. Malmuth, *"Gambling Theory and Other Topics"*, 5th edition, Publisher: Two Plus Two, 1990.

[68] G. Hansen, *"Every Hand Revealed"*, 1st Edition, Publisher: CITADEL, 2009

[69] J. Tao et al, *"Affective computing: A review"*, Volume 3784 of the series Lecture Notes in Computer Science pp 981-995, 2005

[70] B. Browne, *"Going on tilt: Frequent poker players and control"*, Journal of gambling behavior, vol. 5, no. 1, pp 3-21, 1989

[71] G. Smith et al, *"Poker Player Behavior After Big Wins and Big Losses",* Management Science Journal, vol. 55, no. 9, pp. 1547–1555, 2009

[72] R. A. Epstein, *"The Theory of Gambling and Statistical Logic"*, Revised Edition, Publisher: Academic Press Inc, 1995 (Book)

[73] A. Davidson et al, *"Poker Academy Pro - The Ultimate Poker Software"*, Available at http://www.poker-academy.com/ (Online)

[74] L. F. Teófilo, *"Estimating the Probability of Winning for Texas Hold'em Poker Agents"*, Proceedings 6th Doctoral Symposium on Informatics Engineering, 2011, pp. 129–140.

[75] J. Rubin et al, *"Case-based strategies in computer poker",* AI Communications, vol. 25, no. 1, pp. 19–48, 2012

[76] D. Schatzberg, *"Open Meerkat Bot Simulation Testbed"*, Available at http://code.google.com/p/opentestbed/ (Online)

[77] L. F. Teófilo et al, *"Simulation and Performance Assessment of Poker Agents"*, Springer LNCS 7838 (MABS 2012), pp. 69–84, 2013

[78] L. F. Teófilo et al, *"Computing Card Probabilities in Texas Hold'em"*, Proceedings of CISTI'2013 - 8ª Conferência Ibérica de Sistemas e Tecnologias de Informação, pp. 989–994, 2013

[79] L. F. Teófilo et al, *"Speeding-up Poker Game Abstraction Computation: Average Rank Strength"*, Proceedings of Computer Poker and Imperfect Information: Papers from the AAAI 2013 Workshop, 2013, pp. 59–64.

[80] K. Waugh et al, *"Abstraction pathologies in extensive games"*, Proceedings of AAMAS '13 Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems, pp. 271–278, 2009

[81] J. Reis, *"A GPU implementation of Counterfactual Regret Minimization",* Master Thesis, University of Porto, 2015

[82] J. Campos, *"A Profitable Online Poker Agent"*, Master Thesis, University of Porto, 2013.

Appendices

Appendix A List of Publications

During the research that led to the writing of this thesis, several scientific publications were made, which are listed below grouped by type of publication (conference paper, journal paper, book, book chapter or talk):

- **Book Chapters**
  - Luís Filipe Teófilo, Luís Paulo Reis. "Building a No Limit Texas Hold'em Poker Agent Based on Game Logs Using Supervised Learning". Springer LNAI Vol. 6752 – AIS, 2011, pp 73-82
  - Paulo Martins, Luís Paulo Reis, Luís Filipe Teófilo. "Poker vision: playing cards and chips identification based on image processing". Springer LNCS Vol. 6669 – IBPRIA, 2011, pp 436-443
  - Luís Filipe Teófilo, Nuno Passos, Luís Paulo Reis, Henrique Lopes Cardoso. "Adapting Strategies to Opponent Models in Incomplete Information Games: A Reinforcement Learning Approach for Poker". AIS 2012, Springer LNAI Vol. 7326 – AIS, 2012, pp 220-227
  - Luís Filipe Teófilo, Rosaldo Rossetti, Luís Paulo Reis, Henrique Lopes Cardoso, Pedro Alves Nogueira. "Simulation and performance assessment of Poker Agents". Springer LNAI Vol. 7838 – MABS, 2013, pp 69-84
  - Luís Paulo Reis, Pedro Mendes, Luís Filipe Teófilo. "High-Level Language to Build Poker Agents". Springer AISC Vol. 206 – WC, 2013, pp 643-654

— **Books**

- Luís Filipe Teófilo. “Programming a Texas Hold'em Poker AI”. Lap Lambert, 2013, ISBN 978-3-659-45444-8

— **Conference Proceedings**

- Luís Filipe Teófilo, Luís Paulo Reis. “HoldemML: A framework to generate No Limit Hold'em Poker agents from human player strategies”. Proceedings of the 6th Iberian Conference on Information Systems and Technologies, IEEE, 2011, pp 972-977
- Luís Filipe Teófilo, “Estimating the Probability of Winning for Texas Hold’em Poker Agents”. Proceedings of the 6th Doctoral Symposium on Informatics Engineering, 2011, pp 129-140
- Luís Filipe Teófilo, Luís Paulo Reis. “Identifying Player's Strategies in No Limit Texas Hold'em Poker through the Analysis of Individual Moves”. Proceedings of the 15th Portuguese Conference on Artificial Intelligence (EPIA), APIA, 2011, pp 70-83
- Luís Filipe Teófilo, Rosaldo Rossetti, Luís Paulo Reis, Henrique Lopes Cardoso. “A Simulation System to Support Computer Poker Research”. Proceedings of MABS'12 - 13th International Workshop on Multi-Agent Based Simulation at AAMAS, 2012, pp 81-92
- Luís Filipe Teófilo, Luís Paulo Reis, Henrique Lopes Cardoso. “Computer Poker Research at LIACC”. Proceedings of the AAAI Workshops – Computer Poker Symposium, AAAI, 2012
- Luís Filipe Teófilo, Luís Paulo Reis, Henrique Lopes Cardoso. “Computing card probabilities in Texas Hold’em”. Proceedings of the 8th Iberian Conference on Information Systems and Technologies, IEEE, 2013, pp 988-993
- Luís Filipe Teófilo, Luís Paulo Reis, Henrique Lopes Cardoso. “Estimating the odds for Texas Hold'em Poker Agents”. Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT), ACM, 2013, pp 353-360

- Luís Filipe Teófilo, Luís Paulo Reis, Henrique Lopes Cardoso. “A Poker Game Description Language”. Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT), ACM, 2013, pp 369-374
- Luís Filipe Teófilo, Luís Paulo Reis. “Speeding-up Poker Game Abstraction Computation: Average Rank Strength”. Proceedings of the AAAI Workshops – Computer Poker Workshop, AAAI, 2013, pp 59-64
- Luís Filipe Teófilo, Luís Paulo Reis, Henrique Lopes Cardoso. “A Profitable Online No-Limit Poker Playing Agent”. Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT), ACM, 2014, pp 286-293

— **Journals**

- Luís Filipe Teófilo, Luís Paulo Reis, Henrique Lopes Cardoso. “Rule based strategies for large extensive-form games: A specification language for No-Limit Texas Hold’em agents”, Journal Computer Science and Information Systems (COMSIS), 2014, Volume 11, Issue 4, pp 1249-1269
- Luís Filipe Teófilo, Luís Paulo Reis, Henrique Lopes Cardoso. “A General Poker Game Playing System”, Journal Web Intelligence and Agent Systems: An International Journal (WIAS), 2015 **(invited and submitted)**

— **Supervised MSc Thesis (that made contributions for this thesis)**

- Paulo Sérgio Ribeiro Martins. PokerVision - Perception Layer for a Human-Robot Poker Table. Master Thesis, University of Porto, 2011
- Nuno Miguel da Silva Passos. Poker Learner: Reinforcement Learning Applied to Texas Hold'em Poker. Master Thesis, University of Porto, 2012

- João Castro Correia, PGDL: Sistema para definição genérica de jogos de Poker. Master Thesis, University of Porto, 2013
- João Almeida Campos, A Profitable Online Poker Agent. Master Thesis, University of Porto, 2013
- José Pedro Marques, Reinforcement Learning applied to non-deterministic games, University of Porto, 2013
- Carlos Eduardo Frias, Developing an API to assess Poker agents online, Master Thesis, University of Porto, 2014
- Nuno José Velho, Implementação iterativa do algoritmo Counterfactual Regret Minimization, Master Thesis, University of Porto, 2015
- João Reis, A GPU implementation of Counterfactual Regret Minimization, University of Porto, 2015

— **Talks**

- Luís Filipe Teófilo. “Strategies in stochastic games with imperfect information: No Limit Texas Hold'em Poker”. Invited talk at Sapo Codebits, 2010, Lisbon, Available Online at http://videos.sapo.pt/L0gobSW54S3bK7rokoSq

Appendix B Glossary of Poker Terms

This glossary of poker terms was taken and adapted from the book “The Theory of Poker” by David Sklansky [16].

- **Ante**. A fee that is deposited in the pot before the game starts.
- **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.
- **Bankroll**. The amount of money that the player has allocated to the game.
- **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.

— **Check**. To decline to put money in the pot in 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 op in Hold'em).

— **Made Hand**. A hand with a good chance of currently being the best, such as top pair on the op 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.

— **Muck**. On the showdown the players have the possibility of hiding the content of their hands and thus forfeiting the hand.

— **No-Limit**. A poker variant that does not impose a limit on the value of raise actions.

— **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 op 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.

— **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.

Appendix C PGDL Documents

C.1 Kuhn in PGDL

```
<PGDL>
 <PokerGame name="Kuhn" wildCards="false"
winningType="high" ante="1" />
 <History />
 <Description />
 <Players minimun="2" maximum="2" />
 <Deck standard="false" jokers="0">
  <Card id="jh" name="Jack Hearts" rank="0"
suit="hearts" wild="false" />
  <Card id="qh" name="Queen Hearts" rank="1"
suit="hearts" wild="false" />
  <Card id="kh" name="King Hearts" rank="2"
suit="hearts" wild="false" />
 </Deck>
 <Scoring standard="true" handSize="1" />
 <Round number="1" name="Round One"
communityCardsNumber="0" faceUpCardsDealt="0"
faceDownCardsDealt="1" forceBet="false"
blinds="false">
  <BettingStructure type="limit">
   <Bet value="1" maxNumRaises="1" />
  </BettingStructure>
  <DrawCards min="0" max="0" />
  <PassCards value="0" direction="clockwise" />
  <DiscardCards value="0" />
  <BettingOrder order="Clockwise"
firstPlayerBetting="nextDealer" />
 </Round>
</PGDL>
```

C.2 Leduc Hold'em in PGDL

```
<PGDL>
 <PokerGame name="Leduc Hold'em"
wildCards="false" winningType="high" ante="1"
/>
 <History />
 <Description />
 <Players minimum="2" maximum="2" />
 <Deck standard="false" jokers="0">
  <Card id="jh" name="Jack Hearts" rank="0"
suit="hearts" wild="false" />
  <Card id="qh" name="Queen Hearts" rank="1"
suit="hearts" wild="false" />
  <Card id="kh" name="King Hearts" rank="2"
suit="hearts" wild="false" />
```

```
  <Card id="jc" name="Jack Clubs" rank="0"
suit="clubs" wild="false" />
  <Card id="qc" name="Queen Clubs" rank="1"
suit="clubs" wild="false" />
  <Card id="kc" name="King Clubs" rank="2"
suit="clubs" wild="false" />
 </Deck>
 <Scoring standard="true" handSize="2" />
 <Round number="1" name="Round One"
communityCardsNumber="0" faceUpCardsDealt="0"
faceDownCardsDealt="1" forceBet="false"
blinds="false">
  <BettingStructure type="limit">
   <Bet value="1" maxNumRaises="1" />
  </BettingStructure>
  <DrawCards min="0" max="0" />
  <PassCards value="0" direction="clockwise" />
  <DiscardCards value="0" />
  <BettingOrder order="clockwise"
firstPlayerBetting="nextDealer" />
 </Round>
 <Round number="2" name="Round Two"
communityCardsNumber="1" faceUpCardsDealt="0"
faceDownCardsDealt="0" forceBet="false"
blinds="false">
  <BettingStructure type="limit">
   <Bet value="1" maxNumRaises="1" />
  </BettingStructure>
  <DrawCards min="0" max="0" />
  <PassCards value="0" direction="clockwise" />
  <DiscardCards value="0" />
  <BettingOrder order="clockwise"
firstPlayerBetting="nextDealer" />
 </Round>
</PGDL>
```

C.3 No Limit Texas Hold'em in PGDL

```
<PGDL>
 <PokerGame name="No-Limit Hold'em"
wildCards="false" winningType="high" ante="0"
/>
 <History />
 <Description />
 <Players minimum="2" maximum="9" />
 <Deck standard="true" jokers="0" />
 <Scoring standard="true" handSize="5" />
 <Round number="1" name="Pre-Flop"
communityCardsNumber="0" faceUpCardsDealt="0"
faceDownCardsDealt="2" forceBet="false"
blinds="true">
```

```
  <BettingStructure type="no-limit">
   <Bet value="1" maxNumRaises="4" />
  </BettingStructure>
  <DrawCards min="0" max="0" />
  <PassCards value="0" direction="clockwise" />
  <DiscardCards value="0" />
  <BettingOrder order="clockwise"
firstPlayerBetting="nextDealer" />
  <BlindStructure>
   <Blind id="smallBlind" value="0.5"
name="Small-Blind" position="nextDealer" />
   <Blind id="bigBlind" value="1" name="Big-
Blind" position="nextSmallBlind" />
  </BlindStructure>
 </Round>
 <Round number="2" name="Flop"
communityCardsNumber="3" faceUpCardsDealt="0"
faceDownCardsDealt="0" forceBet="false"
blinds="false">
  <BettingStructure type="no-limit">
   <Bet value="1" maxNumRaises="4" />
  </BettingStructure>
  <DrawCards min="0" max="0" />
  <PassCards value="0" direction="clockwise" />
  <DiscardCards value="0" />
  <BettingOrder order="clockwise"
firstPlayerBetting="nextDealer" />
 </Round>
 <Round number="3" name="Turn"
communityCardsNumber="1" faceUpCardsDealt="0"
faceDownCardsDealt="0" forceBet="false"
blinds="false">
  <BettingStructure type="no-limit">
   <Bet value="1" maxNumRaises="4" />
  </BettingStructure>
  <DrawCards min="0" max="0" />
  <PassCards value="0" direction="clockwise" />
  <DiscardCards value="0" />
  <BettingOrder order="clockwise"
firstPlayerBetting="nextDealer" />
 </Round>
 <Round number="4" name="River"
communityCardsNumber="1" faceUpCardsDealt="0"
faceDownCardsDealt="0" forceBet="false"
blinds="false">
  <BettingStructure type="no-limit">
   <Bet value="1" maxNumRaises="4" />
  </BettingStructure>
  <DrawCards min="0" max="0" />
  <PassCards value="0" direction="clockwise" />
  <DiscardCards value="0" />
```

```
  <BettingOrder order="clockwise"
firstPlayerBetting="nextDealer" />
 </Round>
</PGDL>
```

C.4 5 Card Draw in PGDL

```
<PGDL>
 <PokerGame name="Five Card Draw"
wildCards="false" winningType="high" ante="1"
/>
 <History />
 <Description />
 <Players minimum="2" maximum="6" />
 <Deck standard="true" jokers="0" />
 <Scoring standard="true" handSize="5" />
 <Round number="1" name="Round One"
communityCardsNumber="0" faceUpCardsDealt="0"
faceDownCardsDealt="5" forceBet="false"
blinds="false">
  <BettingStructure type="limit">
   <Bet value="1" maxNumRaises="4" />
  </BettingStructure>
  <DrawCards min="0" max="0" />
  <PassCards value="0" direction="clockwise" />
  <DiscardCards value="0" />
  <BettingOrder order="clockwise"
firstPlayerBetting="nextDealer" />
 </Round>
 <Round number="2" name="Round Two"
communityCardsNumber="0" faceUpCardsDealt="0"
faceDownCardsDealt="0" forceBet="false"
blinds="false">
  <BettingStructure type="no-limit">
   <Bet value="1" maxNumRaises="4" />
  </BettingStructure>
  <DrawCards min="0" max="3" />
  <PassCards value="0" direction="clockwise" />
  <DiscardCards value="0" />
  <BettingOrder order="clockwise"
firstPlayerBetting="nextDealer" />
 </Round>
</PGDL>
```

C.5 Custom Scoring Example

```
<Scoring standard="false" handSize="5">
 <Score name="high card" rank="0"
default="true" sort="true" >
  <Subrank>
   $c5.rank * 28561 + $c4.rank * 2197 +
$c3.rank * 169 + $c2.rank * 13 + $c1.rank
  </Subrank>
 </Score>
 <Score name="pair" rank="1" default="false"
sort="true">
  <Conditions>
  </Conditions>
  <Subrank>
   $c5.rank == $c4.rank?
    $c5.rank * 100000 + $c3.rank * 169 +
$c2.rank * 13 + $c1.rank:
   $c4.rank == $c3.rank?
    $c4.rank * 100000 + $c5.rank * 169 +
$c2.rank * 13 + $c1.rank:
   $c3.rank == $c2.rank?
    $c3.rank * 100000 + $c5.rank * 169 +
$c4.rank * 13 + $c1.rank
    $c2.rank * 100000 + $c5.rank * 169 +
$c4.rank * 13 + $c3.rank
  </Subrank>
 </Score>
 <Score name="two pairs" rank="2"
default="false" sort="true" >
  ...
 </Score>
 ...
</Scoring>
```