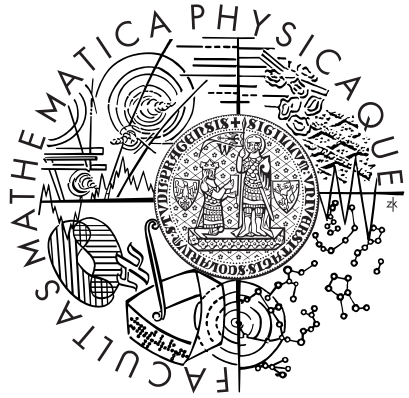
Charles University in Prague
Faculty of Mathematics and Physics

MASTER THESIS



Pavol Müller

Prostředí pro vytváření umělých hráčů Texas Hold'em pokru

Environment for creating automated Texas Hold'em poker players

Department of Theoretical Computer Science and Mathematical Logic

Supervisor: Mgr. Vladan Majerech, Dr.

Study Program: Computer Science

I would like to express my gratitude to all those who helped me complete this thesis, especially to Miroslav Kolínsky for his valuable comments and corrections and to Mgr. et Mgr. Evžen Bouřa, PhD. for the consultations and help with the formalization and implementation of the poker algorithm. My thanks also go to Mgr. Vladan Majerech, Dr. for his guidance and leading of this thesis. I dedicate this work to my parents who have supported me throughout my studies.

I declare that I wrote this thesis on my own, using only the stated resources. I agree that the thesis can be made publicly available.

Prague, April 17, 2009 Pavol Müller

Contents

Názov práce: Prostředí pro vytváření umělých hráčů Texas Hold'em pokru
Autor: Pavol Müller
Katedra: Katedra teoretické informatiky a matematické logiky
Vedúci diplomovej práce: Mgr. Vladan Majerech, Dr.
e-mail vedúceho: Vladan.Majerech@mff.cuni.cz

Abstrakt: Hold 'em poker, kartová hra s neúplnou, niekedy dokonca klamlivou informáciou, bude v priebehu niekoľkých rokov pravdepodobne pokorená počítačovými programami. Toto môže predstavovať veľké nebezpečenstvo pre celé odvetvie on-line pokeru. V niektorých variantoch Texas hold 'em pokeru sa tak už deje. Táto práca skúma možnosti vytvorenia automatického hráča no-limit turnajov, ako najmenej preskúmanej a najťažšej varianty Texas hold 'em pokeru pre počítačového hráča. Budú vytvorené nástroje a metodológia, ktoré umožnia obecnú a jednoduchú extrakciu dát z on-line kasín založenú na optickom rozpoznávaní. Automatické operácie v týchto kasínach a prostredie na programovanie pokerovej inteligencie budú taktiež zahrnuté. Následne bude implementovaný umelý pokerový hráč schopný hry v dvoch kasínach, ako ukážka robustnosti a použiteľnosti tohto prostredia. Budú predstavené známe pokerové stratégie a myšlienky z pohľadu implementácie pokerového robota. Na konci bude tento umelý pokrový hráč odskúšaný na reálnych situáciách a na základe získaných skúseností a výsledkov budú zodpovedané otázky týkajúce sa zneužitia on-line kasín umelými pokerovými hráčmi.

Kľúčové slová: poker, bot, robot, kasíno

Title: Environment for creating automated Texas Hold'em poker players
Author: Pavol Müller
Department: Department of Theoretical Computer Science and Mathematical Logic
Supervisor: Mgr. Vladan Majerech, Dr.
Supervisor's e-mail address: Vladan.Majerech@mff.cuni.cz

Abstract: Hold 'em poker, a card game with incomplete, sometimes even misleading information, is probably going to be mastered by computer programs within a few years. This might be a big issue for the on-line poker gaming industry. In some variations of the Texas hold 'em poker it is already happening. This thesis explores a possibility of creating an automated no-limit tournament player for the less explored and the hardest variation of Texas hold 'em poker from a computer player point of view. Tools and methodology which allows universal and simple automated data extraction from on-line casinos based on optical recognition are going to be created. Automated operations in these casinos and environment for poker intelligence implementation will be included as well. An artificial poker player capable of playing in two casinos will be implemented in this environment to demonstrate its strength and usability. Known poker strategies and ideas are going to be introduced from the point of view of poker robot implementation. At the end, the automated poker player will be confronted in real situations and questions regarding misuse of automated poker players in on-line casinos based on gained experiences and results are going to be answered.

Keywords: poker, bot, robot, casino

Preface

I, just as many people nowadays, was one of those who were fascinated by a phenomenal card game that has been spreading over the world during the last decade. A game that already managed to throw a shadow on chess and blackjack. A probability game that honors logic and skill, with a significant aspect of psychology and without any explicit winning strategies. A game with simple rules and endless possibilities where money are only at the first place. The Texas hold 'em poker.

I have became an occasional on-line poker player, like millions of others. Some of them got to be more or less profitable and some of those profitable surely got a pub-idea of creating a robot that would play on-line instead of them according to their style and tactics. Of all these people there must be some of the same profession as I am and do the same thing as I did. To start putting together some after-work thoughts and ideas on how to build that robot. As a possible path started to reveal itself, I figured out that I am already behind. Also, another questions arises here. Is it possible that we are already playing against robots on-line? Is it possible that there are machines already sucking up money out of regular players? How difficult would it be to build that robot? What variants of poker games could this robot play? Which casinos could be vulnerable?

I have therefore decided to put my academic task and my hobby together and answer all of pronounced questions in this master thesis. These questions will be answered through research and development of an environment used for creating automated on-line poker casino analyzers and players.

Thesis objectives

To be able to answer these questions, and maybe even more, we have determined a few objectives for this thesis.

- The primary goal of this thesis is to build an environment that allows construction of automated poker players capable of playing in most of currently available casinos.

Hand in hand with the primary objective goes the secondary objective.

- Use this environment to create an artificial intelligence that can eventually play in one of these real casinos.

If we succeed in building that kind of environment and we will be able to use it to create a player that could play in a real casino we may try to investigate our third goal.

- Analyze the possibility of creating a profitable artificial intelligence capable of playing hold 'em poker automatically.

By fulfilling or failing to achieve these objectives we should be able to answer every question we have asked ourselves in the beginning.

This thesis is (not)

- It is not another guide on how to become a better poker player, however some approaches that work for an algorithm may help the reader to realize some mistakes and adjust his or her game play. Everything is considered from a point of view of an automated player.
- The result of this thesis will not be a free money making tool. Even if we are lucky enough to succeed in our implementation, we will not and cannot publish all the results because of understandable reasons and consequences.
- Using certain results of this thesis might be illegal in some countries or against the rules of some casinos and considered as cheating. A person violating these rules may put himself at risk and face appropriate consequences. The aim of this work is to answer the defined goals without breaking any low or casino agreement rule.
- This thesis is designed only for theoretical and academics purposes. Using this thesis or any part of it in private or with any private or commercial intention is not allowed.

Chapter 1

Introduction

In this chapter we are going to intimate the reader with the basic rules, terms and overview of Texas hold 'em poker necessary for further understanding of the thesis. If the reader is familiar with the situation of on-line poker gaming and feel comfortable with terms like blinds, flop, outs or pot odds and know the difference between a ring game and a tournament then he can feel free to skip this chapter. Besides, if an uncanny term occurs without appropriate explanation, it is possible to look it up in the glossary post facto.

1.1 On-line gaming overview

There were two huge turnovers in hold 'em poker during the last few decades that utterly influenced the popularity of this game. First, the advent of minicams that allows TV viewers to see the hands of professional players right as they were making their decisions. People were now able to understand some tactics and realized the beauty and complexity of the game. The second milestone came with the expansion of the Internet and introduction of the on-line Internet casinos. Suddenly the game became open for mases, not only for a selected few who could afford the high stakes in real casinos.

These days there are more than a hundred of these virtual Internet casinos. They are operating with real money in the most widespread currencies. It is up to the user to decide for a game that best suits his or her potential and financial situation. The casinos offer games with stakes as low as a few dollar cents up to few hundreds dollars per one hand. The overall price pool in dollars of some occasions can reach seven digits. Supported by the simplicity of on-line payments, the amount of money in this sector is still rising despite many government interventions already done against this industry.

Getting a significant portion of this money is not as easy as one could think.

Although the rules of Texas hold 'em poker are very simple, the game complexity and factors that can influence in-game decision making are hidden to many casual players. This is very good for us and our prospective bot. On the other hand the fees (called rakes) for the casino are very high. They vary somewhere around 10%. We will need to get beyond this barrier with our play to became permanently profitable. This is something that only one of ten players playing on-line is able to achieve in the long run. Luckily, there are some ways to reduce these expenses.

1.1.1 The costs and revenues

The casinos put a lot of money to advertisement and promoting actions. Almost all of them are offering the so-called first deposit bonus. If a player starts playing on a poker site, they will offer him additional bonus between 100 to 150% of his first deposit, on some sites up to $1000. This is of course conditioned with a requirement of playing a specified number of hands or in other words paying enough on fees so they could give the player something in return. Not bad either, but could be applied only once per person and casino.

Another way how to get permanent rake discounts is by using one of the rake-back sites[1]. Some casinos offer a permanent payment to web pages that advertise their poker site. When a player registers through this advertisement and starts spending his or her money in the casino they share a part of the fees with the web site. The point is now clear – there are sites that share a portion of their part with a player who registers through their poker advertisement link. This is a significant remittance, somewhere up to 30% of the fees paid to the casino.

The last alleviation of the impact of rakes could be done through bonus programs of each casino. One can order free items and gadgets from the poker store, the value of the goods is derived from the amount of money the player spends in the casino. It is possible to choose items ranging from poker books, tickets to a variety of poker events and tournaments, clothing, electronics or even up to Porsche Cayman for example.

1.2 Texas Hold 'em poker

This game can be characterized as a social, probability card game with incomplete information. But unlike a roulette or a poker on a gaming machine it is not a gamble where a player is condemned to loose in the long term. Here he or she competes only with other players with exactly the same rules on all sides. In a lot of cases somebody may get lucky and hit a very improbable combination that makes others

[1]For example the *Rakeback.com* at *http://www.rakeback.com/* or the *Rakeback lowers* at *http://rakebacklovers.com/.*

loose a lot of money, but everything must be contemplated from the long point of view. This is what Texas hold 'em is all about. Making the right decisions that are profitable in the long run despite that sometimes it may hurt.

1.2.1 General rules

This poker is played with a standard deck of 52 playing cards. Thirteen cards from deuce to ace in all four suits: clubs, diamonds, hearts and spades. There may be from two up to ten players around a table playing clockwise. Every player gets his own two playing cards. Five more public cards are subsequently revealed. This is interlarded with betting and stepping aside of some players. Finally, the last man standing takes the money or a showdown distinguish the winner according to the best five cards picked from the two private and five public cards for each player.

1.2.2 The very first hand

Dealer position is picked by a lot at the beginning and shifts every hand one place to the left. Before the dealer deals two hidden cards to all players two obligatory bets are done by the two players sitting on the left of the dealer. The size of these bets is predetermined by the game type, the player closer to the dealer puts in half of the bet of the next player. These bets are called *small blind* and *big blind*.

After everyone takes a look at his or her cards, the first round of betting takes place starting with the player to the left of the big blind. Players act subsequently and everyone has one of the following choices: step aside by *fold*ing the hand, *call* and toss in the actual bet (which equals to the big blind at the beginning of the betting round) or *raise* the actual bet. Raising must reach or exceed the minimum allowed amount which is the maximum of either the big blind or the last raise in this betting round. The betting stops at the position of the big blind when everybody folded or called the highest raise. The big blind might also *check* when nobody raises. This means he decides to stay in the game without additional raising.

In case that size of a raise is higher than a player can afford to bet, he doesn't need to fold. He might choose to bet all his remaining chips and play for an adequate portion of the chips to the end with all players that remains in. This kind of move is called an *all-in*.

After the initial betting took place, the first three common cards are dealt face up on the table. (This is called the flop.) Every player now has a hand consisting of five cards with the possibility of improvement with the next two public cards that are going to be revealed. But at first, another round of betting takes place. This time starting at small blind position and ending at dealer position. Subsequently another two face up cards land on the table interlaced with a round of betting after each

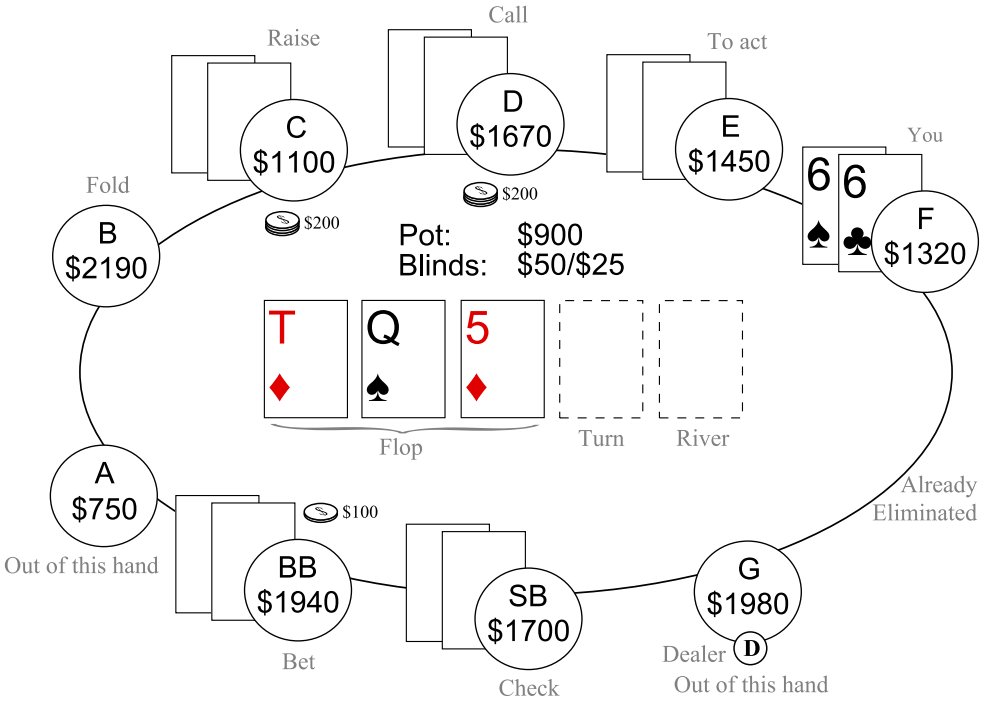


Figure 1.1: Poker table example

card. These cards are called the turn and the river. The betting is done the same way as after the flop. An exemplary post-flop betting situation is depicted in Figure 1.1.

The concluding showdown determines the winner considering the best five cards out of corresponding seven. In case of draw the prize splits.

1.2.3 Hand strength

Following hands are ranked from the weakest to the strongest. A corresponding example is provided in parenthesis. We will use "T" for tens. Later we will use "s" for suit indication. For instance a hand ATs means ace-ten suited. A card represented as "x" means any card.

- High card (K♦)
- Pair (7♥7♣)
- Two Pair (4♠4♦ T♥T♠)
- Three of a kind (J♥J♠J♣ Also called a set.)

- Straight (8♣9♠T♣J♥Q♦ Five cards in a sequence, ace may be used as one in front of a deuce.)
- Flush (2♦9♦Q♦7♦A♦ Five cards of the same suit.)
- Full House (K♠K♦K♣ 2♥2♦ Three of a kind with another pair.)
- Four of a Kind (6♥6♦6♣6♠)
- Straight Flush (A♥2♥3♥4♥5♥ Five cards in a sequence in the same suit. Ace may be used as one.)

If a situation occurs where players have a hand of the same strength, the highest lower sub-hand distinguishes the winner. For example, if both players have a full house the one with three of a kind of higher value wins. If the players share the set then the pair is taken into account. In case that both players have a flush or a straight the higher card gains the prize.

1.2.4 Game variations

There are three variations and two ways of playing Texas hold 'em poker. Although the difference in the mentioned variations is only in maximal allowed bet, the consequences for the game-play and strategy are vast. To be specific, the three versions are the *fixed-limit*, *pot-limit* and *no-limit*. In the fixed-limit variation, the sizes of bets and raises are determined, meaning a player can only decide to raise, but cannot influence the size of the raise. The pot-limit variant has a betting cap on the amount that can be bet and this cap is equal to the size of all bets made until the actual hand. The last and most complicated is the version without any limits where a player can commit everything he has any time he wants.

In addition there are two approaches on how to play these variations – a ring game (also called a cash game) and a tournament. In the first one the bets represent real money. The objective is to accumulate money faster than loosing it. In the second variant an entering fee is being paid and everybody gets the same amount of virtual jettons called chips. The objective is to stay in the game as long as possible, preferably to be the last one with chips. In tournaments with nine to ten players the first three places are usually paid with rates of 50, 30 and 20 percent of all entering fees. To ensure that the tournament finishes in a reasonable time, the blinds are rising regularly.

1.2.5 Glossary

- **Ante**: An obligatory payment made by all players before the cards are dealt. It is markedly smaller compared to the blinds and should boost up action

especially in later phases of tournaments.

- **All-in**: Betting all chips and playing the hand to the end only for an adequate portion of the chips.
- **Big blind**: Initial obligatory bet made by the player sitting two places to the left of the dealer. This term can also refer to a player sitting on this position.
- **Bubble**: Phase of a tournament when there is only one unrewarded place left.
- **Button**: Player in the position of a dealer.
- **Covered, be**: A player is covered when he might be eliminated from a tournament by his opponent when going all-in, but not vice versa.
- **Fifth street**: The fifth card revealed to all players after the third round of betting.
- **Flop**: Three cards revealed to all players after the first round of betting.
- **Fourth street**: The fourth card revealed to all players after the second round of betting.
- **Hand**: The two cards that a player is holding or the best five cards of a player, depending on the context.
- **Implied pot odds**: The same as pot odds but also takes into consideration estimated future betting.
- **Kicker**: A card that determines the winner in case of the same hand rank.
- **Limp**: To call a bet of the size of a big blind with intention to see the flop cheaply.
- **Loose**: A player or style is referred as loose when often entering the pot and playing many hands.
- **Out**: A card that improves the hand to the winning hand.
- **Pot**: The overall amount of chips bet by all players in the current hand.
- **Pot odds**: The ratio of the actual pot and the amount of chips necessary to call a bet.
- **River**: The same as the fifth street.

- **Slow-play**: Not betting with a strong hand with intention to let other(s) take the lead in betting.
- **Small blind**: An initial obligatory bet made by the player on the left of the dealer. This term can also refer to a player sitting on this position.
- **Stack**: The overall amount of chips for each player.
- **Tight**: A player or style is referred as tight when rarely entering the pot and playing a few hands.
- **Turn**: The same as the fourth street.

Chapter 2

Data mining

In this chapter we will summarize all the available information related to our thesis' objectives. We will take a survey of currently operating casinos, analyze the practicability of our ideas according to given conditions and take a look on other works and programs concerning our targets. As a result of these investigations we will try to formulate some definite requirements for further analysis and software design.

2.1 Poker sites in general

The term poker site or a poker room refers to a server – client based architecture. A central server, or a group of servers, manages the poker tables and handles the requests from players' clients. These clients are executable applications, mostly based on flash or java technology, designed for systems running MS Windows. However, there are a few sites with mac support or at least with an announcement that the mac version of their software is on the way. User can interact with the clients in a standard way – using both a mouse and a keyboard to perform an action based on the situation displayed in the client. The communication between the poker server and a locally installed client is encrypted. There are no data that can be gathered by observing the tcp/ip communication.

- Our application's target platform will be MS Windows.

Poker sites use either their own software or one of already existing software customized to their preferences. This customized software may share users among different poker sites, creating a poker network like Merge, CryptoLogic or IPN. The biggest sites usually use their own poker software.

Surprisingly, the difference in quality of individual poker clients is enormous. We can find perfect bug-free applications providing all the necessary information we need, but at the same time there are applications that crash very often or applications

drawing black texts on black background, for example. Playing on such sites is not only impossible, but makes the choice of the right casino easier when considering that this software has access to information about our credit card.

2.1.1 Casino agreement rules and third party tools

Despite the fact that casinos profit on every player as long as he plays, they made a strict stand against the usage of certain supportive tools and software, automated players included. This might be caused by a possible loss of active human players if the tables were full of robots squeezing money form beginners and inexperienced players. After all, nobody leaves an unprofitable robot playing in a casino in the long term. Moreover, considering the casino fees, even the elite robots cannot make money one from another.

The line between good and bad, or rather allowed and disallowed, differs among poker sites. Generally they did not mind if a player educates himself using any source of literature and learns some tactics and math to improve his playing style. Even some software that does the math and computes things like outs, probability of hitting a card etc. is mostly tolerated. However, it must not make any suggestions about the action that the player should make. Interacting with the poker client is usually allowed only to humans. This is quite a trouble for our objective, but fortunately there are also sites that allow automatic interaction. Then again, the automatic interaction is conditioned by human activity. An example of an allowed tool might be a software that bets the size of the pot when a hotkey is hit by a user. On the other hand the bad side can be represented by an automatic folder which automatically folds all useless hands. One thing on which almost all sites agreed is the cooperation and usage of game data that was collected by other players – not allowed. Tools that profile player's opponents using the information they have accumulated are generally allowed and a player can take advantage of these data if he wants. There is a third category of tools that might be used, but not during an on-line play. To this category belongs most of the poker training software and software handling shared user statistics.

2.1.2 Rules violation policy

The sites generally declare a zero toleration policy on infringement of their rules. On many sites it is not only a declaration but rather concrete steps trying to detect any misuse of their site. From among the the most common techniques we mention a scanning of running processes, searching for installations of prohibited software or even monitoring the web browser for accessing forbidden on-line services, mostly the large shared databases containing poker players and their skills. However, many of these techniques might be fooled with some effort. It is a game

of a cat and a mouse. Therefore another approach appeared – monitoring the user behavior patterns. When there is somebody playing on ten tables 24 hours straight without having a minute off, at leas for a bath break, then there is something suspicious. Some casinos declare that they have up to one hundred employees every day searching for automated players and software providing unfair privileges.

The process after disclosing prohibited software is quite straightforward. In minor offences the users receive a warning email that they are using disallowed software and if they want to continue playing they should stop it or uninstall it. In repeated or more severe delicts the user account is immediately suspended, the funds on the account are confiscated and used as a compensation for affected player. There are also databases among poker sites containing information about players with serious delicti.

This is something where we definitely do not want to found ourselves during the implementation of the goals of this thesis.

2.2 Related software in general

As mentioned already, there is a group of available programs that have something in common with poker or even with automated poker playing. In this sub-chapter we will make a quick overview of such applications and try to poke out the general pros and cons.

2.2.1 Data analyzers and profilers

This group of software is designed to collect data from real game play. Most of these tools uses the log processing approach. Logging the user actions into a text file is supported by most of the casino clients. After playing a significant number of hands a player may get a notion on some actions he did with these programs. For example, he can see the overall profit when calling or raising with a specific hand and he may adjust his play according to these observations. But we have to keep in mind that sharing this data or using other player's data is usually in a conflict with casino usage rules. Such software that natively supports the sharing may be considered as prohibited and checked by the poker application.

2.2.2 Poker calculators

This group of applications usually belongs to the partially allowed group. They can be used "off-line" when not playing on poker sites to analyze some specific situations for the user to understand the difference between the right and wrong decision. But can not be used during the real play to analyze actual hand and situations. The

supported features of this calculators differ from the most fundamental computations like pot odds, hand strength and outs analysis to integrated solutions capable of taking into account many advanced approaches.

2.2.3 Existing poker robots

The third group of related poker software are the automated advisors or even automated players. Using this group is of course strictly forbidden. This is also the category in which we are very interested in this thesis and that we are going to analyze the most. We will just mention the biggest issues concerning these bots here, the deeper analysis will follow in the next subchapters.

Some of these tools are based on monitoring graphical representation of a poker table and screen recognition, some of them use specific methods to extract the required information from the client. Some of them even emulate user interaction with the poker client using mouse and keyboard or a any other specific approach if the appropriate client allows. Another thing that almost all of this robots have in common is that they do not work at all. Or even it they are capable of making some profit it is only on specific game types. It is only a question of time when the number of robots on these specific games exceed a breaking point and the last profitable concerned member will be the casino. Another question of time is when the bot will be detected and banned by the hosting poker site. This applies to every single bot which was made public or revealed.

At this point we should mention the first and probably the most widespread autoplaying poker software used nowadays - the WinHoldEm. It is configured with a set of yes/no formulas for every hand through a built-in dialogues with a basic C-like operator syntax. It has been designed to play the fixed-limit tournaments according to these formulas, although it can be more or less configured to play even the different game variations. To start playing using the WinHoldEm, the bot must be seated on a table by the user.

2.2.3.1 How do they "work"?

In case of the advisors, the contribution to the improvement of players is straightforward. Providing a real-time deep through analysis for the actual hand is an advantage that no human is capable of making. The calculated results might be a very strong weapon for a skilled player.

The benefit of a fully automated player is clear as well: earning money without any need of user time. Here we can see there different ideological approaches:

- A bot that works is made. The author keeps it for himself and enjoys the money.

- A bot is made, does not matter whether it works or not. The author tries to sell it to as many people as possible before it is banned on most of the supported casinos or before there are so many users using the bot that it stops working. According to the user experiences observed at discussion forums, this was usually the case when the bot was not working at all from the beginning.
- A free bot is made, so people can program it to master the play. This ensures constant development and shifts in the tactics and behavior, while potentially allowing the bot to work for a while. The tricky part is that with expansion of the bot it would be possible to collect endless data and user statistics, based on which a bot mentioned in the first bullet can be built.

2.2.3.2 Avoiding detection

It might be harder to hide a publicly available bot playing but there are ways of doing it good enough. As mentioned before, poker sites monitor the usage of prohibited software. In case of a public robot we can be sure that the time before it gets on the forbidden list is counted in days on the biggest poker sites. Changing the process name might help, but the authors of those bots usually recommend using two computers with a shared desktop. On the first one we have the poker software and standard applications for desktop sharing installed, on the other we run the disallowed robot or software that makes the decisions and propagates the actions back to the first computer. This approach should work as well when using a virtual system instead of the second computer, for example WMware.

2.2.3.3 What do they generally play?

In one sentence, the bots usually play fixed-limit cash games on the lowest stake levels. A more interesting question should be: why do the bots play mostly these games? The answer is: because the conditions are unchanging and it is easy to compute and calculate probabilities when a player knows sizes of possible bets and raises in all later rounds. The decisions are therefore significantly easier and it is of course markedly cheaper when the bot is fooled by an experienced human opponent. Another argument in favor of these tables is that many beginners start on these tables for the exact same reason. While they figure out the basics, the mathematically correct play of the robot might be enough to generate some profit. Designing a robot for a fixed-limit ring game is also much easier. Declaring hands for entering the pot from all positions and subsequently attaching one of the pot odds calculator to distinguish whether to stay in the pot or get out is an easy task. This approach should probably do the job at the lowest stake fixed-limit tables. There is also a lot of software doing this and it is likely that it might work if there are not many similar bots around the table.

That is why we are going to focus on the more challenging no-limit version of poker in this thesis, as it seems that the question about the robots and fixed-limit games can be answered straight away. As Crandell Addington[1], a member of the Poker Hall of Fame[2] said to internationally syndicated newspaper poker columnist Chuck Blount at the end of year 2007 commenting the possibility of using poker robots on-line:

> *"A computer program cannot be written that will consistently defeat a skilled no-limit player if the game is structured for play."*

Addington said.

> *"No-limit is a dynamic battlefield on which players must make rapid strategic shifts and tactical implementations. Computer programs cannot perceive these shifts."*

Although now in the year 2009 we agree with this statement, by leaving out the word "skilled" from the citation we get a very challenging pronouncement to investigate.

- We will focus on the *no-limit* version of Texas hold 'em poker in this thesis.

Another thing we should clarify at this point is whether we want to focus on ring games or rather on the tournaments, especially the most popular one-table sit and go tournaments. Again, most of the software already available is designed to play cash games, because of the unchanging conditions and game style that does not need to be adjusted in any way during the game. It is much easier to pick up a different table to play on, rather than trying to outplay our opponents if they are good enough to outplay our bot.

- We will focus on the Texas hold 'em poker *tournaments* in this thesis.

After a detailed inspection we found out that some robots declared to play the no-limit version and even the tournaments, but the design usually did not take into account some specific requirements that are essential for these game types. They were usually based on the earlier implementation which supports the fixed-limit version and suffers from lack of provided information. For example, the information on betting in previous rounds is essential for the no-limit, whereas in the fixed-limit version it is mostly a question of offered odds based only on the current situation which consist of the cards and the implied pot odds. In tournaments a player also needs to consider the pay-out structure, rising blinds and changing conditions like the number of his opponents to adjust his play accordingly.

[1]He is a no-limit tournament hold 'em legend and is one of the founders of the World Series of Poker. He still holds the record, seven, for most final table appearances in this event.

[2]The Poker Hall of Fame is a group of poker players who have played poker well against top competition for high stakes over a long period of time. Since it was established in 1979 only 37 players were admitted to this group. It is considered as one of the biggest honors in poker.

2.2.3.4 Poker robots in the academics environment

The poker robots are already being developed in the academic society. Unlike chess, for example, here the programs need to solve problems when they are working with incomplete, even misleading, information. There are actually some poker robot tournaments where the programmers, or rather their robots, compete against each other. The focus of these robots is mostly aimed at the fixed-limit version of poker and the cash games.

At this point we should mention and introduce the most successful research group at the field of poker robots nowadays. The Computer Poker Research Group at The University of Alberta[3]. The main aim of their work is to develop a professional fixed-limit cash game hold 'em player. At professional poker level, a perfect math background is a matter of course, while the ability to read the opponents becomes more important. The robots need to prepare traps and detect tricks of their opponents' playing style. In July 2008 their robot Polaris won the 2nd man-machine showdown for the first time playing against some of the best heads-up fixed-limit players in the world. Although this is a great success in our field of research, generally we are still only at the beginning. Polaris is only capable of playing the fixed-limit cash game against a single opponent.

2.2.3.5 Robots' AI

Implementation of a poker bot intelligence and decision making system might be covered by a countless number of approaches. Many of the mentioned applications offer a method for adjusting their advices and decisions. They vary from the most simple ones, like setting an action for each hand, to the most complex approach including interpreted poker rules definition languages or dll interfaces that could be implemented by the user. Another category could be formed by fixed approaches which are based on given methodology, for example neural networks, genetic algorithms or calculators for specific poker related values, for making a decision. In these cases the user can adjust the algorithm mostly by sliders and values that affect the decision making system.

2.2.4 Summary

Combining all the approaches and information we already have, it is obvious that making an automated poker player is definitely possible. If there are ways of extracting the necessary information from a poker table and emulate user interaction on a poker client it should be possible to attain our goal. Especially when there

[3]More information can be found at *http://www.cs.ualberta.ca/~games/poker/.*

already are robots capable of defeating human players. Considering all the work already done, it is not entirely safe to join an on-line low fixed-limit cash table for an inexperienced player. Playing the no-limit tournaments still seems to be save. We will try to confront this pronouncement in this thesis. Nobody knows how many bots are already out and operational, but there are webs on the Internet where people claim that they have created one.

The impossibility of revealing any statistics about bot penetration of on-line casinos is based especially on the fact that if someone is making profit using a bot, he must stay silent at all costs. The poker sites are trying hard to earn a bot-free reputation in order to satisfy their customers.

We have therefore slightly adjusted our thesis' goals to reflect this situation. We will specialize our environment and prospective poker intelligence to deal with the no-limit tournaments as the less explored versions of hold 'em poker from a robot point of view.

2.3 Players' skills

Generally, a simple rule applies. The higher the stakes, the higher the skills of all participants. At lowest stakes there are players that do not even understand the basic probability principles of the game, and play according to their feelings. They are a good pray, but are often a fate for a player relying on a basic intellect of other players. Just above the smallest tables a perceptible drop-off of flagrant decisions is markable. With higher stakes the psychological aspect of the game-play raises. This is probably an area where a robot cannot catch on. On the other hand, the advantage of a robot is that it will never get angry and make an incorrect move afterwards. Because a bot doesn't get bored playing on the lowest stakes all day long, this should be our target field. The influence of more profound psychological skill is less significant on these low-stake tables.

2.4 Further factual analysis

In this subchapter we will continue the data mining process and go deeper considering particular approaches and related software tools. We will take a closer look at individual on-line casinos and try to determine and analyze possible approaches for building an universal environment for poker bot creation.

2.4.1 Representative casino set

As mentioned already, there are a few poker networks each using similar poker software. We will pick one or two casinos from the most known poker networks as typical representatives for further investigation. The main focus will be aimed at the biggest poker sites of these days:

- PokerStars (*http://www.pokerstars.com/*)
- FullTilt (*http://www.fulltiltpoker.com/*)

Along with them we will take a look on the following popular and well-known sites and a few less noted, just to cover a representative set of nowadays poker sites:

- PartyPoker (*http://www.partypoker.com/*)
- Titan Poker (iPoker network, *http://www.titanpoker.com/*)
- UltimateBet (*http://www.ultimatebet.com/*)
- Absolute Poker (*http://www.absolutepoker.com/*)
- CelebPoker (IPN network, *http://www.celebpoker.com/*)
- William Hill (cryptologic network, *http://www.williamhillpoker.com/*)
- InterPoker (cryptologic network, *http://www.interpoker.com/*)
- Allstar (cake network, *http://www.allstar.com/poker/pokerlobby.php/*)
- RedStar Poker (cake network,*http://www.redstarpoker.com/*)
- CD Poker (iPoker network, *http://www.cdpoker.com/*)
- Paradise Poker (IPN network, *http://www.paradisepoker.com/*)
- Euro Poker (onegame network, *http://www.europoker.com/*)
- RedKings (onegame network, *http://www.redkings.com/*)
- Carbon Poker (merge network, *http://www.carbonpoker.com/*)
- CityPoker (merge network, *http://www.citypoker.com/*)
- Pacific Poker (Pacific network, *http://www.pacificpoker.com/*)
- PokerHills (B2B network, *http://www.pokerhills.com/*)

- 24h Poker (B2B network, *http://www.24hpoker.com/*)

PokerStars is the biggest and the most popular poker site at the moment. It is also the biggest poker playroom of all times. In the evening peaks there are more than 200 000 people playing all kinds of poker. This site has brought up four WSOP[4] champions and many WSOP final table participants. Their software is generally considered as the best on the market. This site is so popular that it does not need to offer any rake-back bonuses and even their first time deposit bonus is less than one tenth of the usual bonus on other sites. They ground on their brand and the VIP programs offering exclusive items and gadgets. An important part of the protection of their brand is the most successful poker bot detection. We have not seen any prohibited tool available on the Internet that can run with PokerStars without the need of using stealth approaches. When a new and suspicious application leaks to the Internet they react within a few days.

FullTilt on the other hand seems to have been a very progressive recently. Although the number of active players is only a third when compared to PokerStars, its popularity is rising. This is mostly because some of their players succeeded at the WSOP well and brought fame to the brand. Moreover, compared to PokerStars, they are offering a rake back bonus and much higher first deposit bonus which attract a lot of newcomers and even some semi-experienced PokerStars players.

The rest of the list is filled with some very popular and high-quality sites like TitanPoker (as a part of the iPoker network) or PartyPoker that deserve to be mentioned. On the other hand there surprisingly are sites with doubtful trustworthiness. The pop-ups on their web pages evoke an impression of webs dedicated to adults and the email provided for registration become filled with endless bonuses and "winning" offers.

2.4.2 Extracting data from a casino

Basically there are two approaches how to figure out what is going on in a casino and at a poker table. Of course, neither of these approaches involves an interface on a poker client allowing robot creation as on some applications designed for poker bot competitions. In fact, both of these ways are confounded. Poker client decomposition, dll injection, component API or system GDI hooking represent the first choice. The second is based on screen capturing and graphical recognition, probably with some kind of OCR. We will assume that the overall idea is clear to the reader and just confront these two approaches with respect to our needs for poker bot creation.

If we suppose that execution of both methods is feasible for a given casino, we get the following pros and cons:

[4]World Series of Poker, more information can be found at *http://www.worldseriesofpoker.com/*.

- Decomposition or at least monitoring of poker client actions and operating system calls is practically always against the product usage agreement rules.
- Direct operation with client is more error proof compared to the graphical analysis. Graphical analysis may suffer from hardware-software redraw issues that sometimes occur, especially on Windows operating machines.
- It is not guaranteed that it is possible to extract all the necessary information with each method. Although in the second case it is only a question of the recognition module and its abilities. We are not considering the casinos where even human player cannot read or understand the scenery.
- Graphical analysis can be designed as a general solution for a broad spectrum of casinos, compared to the individual approach that requires a dissection for each casino.

These two methods should be investigated individually for each casino. We will use our representative set of casinos and analyze some key factors. From the overall point of view and the applicability in this thesis the graphical approach seem to be favorable. Therefore we will focus mainly on the attributes necessary for graphical recognition and try to appraise the burdensomeness of this approach.

2.4.2.1 Poker client decomposition

One possible way how to decompose a poker client has been described at Coding The Wheel[5]. The related article is focused on a poker client decomposition, considering both PokerStars and FullTilt clients. Description of the entire process including difficulties with extraction of information is included. Extraction of the chat log component with dealer messages and spying on the hand history log promptly is taken into account. There are also some examples of working code for inspiration together with precompiled binaries. Using these binaries is not risk-free, as some accounts on PokerStars were already banned due to experiments beyond the acceptable boundary.

2.4.2.2 Graphical recognition

A question crucial for this approach is whether the scene at a poker table is generated dynamically according to actual conditions, or every component of a table has its fixed location. In advance of further investigation we need to disclose that

[5]Coding the Wheel: *How I Built a Working Poker Bot*; 2008, *http://www.codingthewheel.com/archives/how-i-built-a-working-online-poker-bot-4*, see especially sections 4, 5, 6, 7.

it is in our favor. This also makes a sense for a human, nobody wants to figure out whether something is closer to something else and therefore related with it before every action taken. It is important to be able to look at a specific place and see the thing. There are some exceptions, but nothing fatal that could not be handled.

Among other observations we will take a look at the possibility of disabling potentially disturbing animations during the game play. Although this does not indicate a problem, it is necessary to keep in mind that some items might be moving out of their desired positions. Waiting until the animation in finished is the simplest solution.

Another requirement for graphical recognition is the ability to see the required objects. We can not recognize something that is overlaped with anything else. The simplest solution could be a window manager which ensures that each window, especially a poker table window, has its own respective position. This also eliminates problems related to sudden window order change during recognition or slightly before the appropriate screen shot is taken. If we want to operate on more tables simultaneously, we will appreciate poker windows that can be miniaturized. This is yet another attribute to observe.

Manipulation with system and poker windows can be implemented using system resources. A different approach could include MS Windows window handler as a part of our graphical recognition environment. In that case we might want to find out whether the poker server use a standard MS Windows windows or its own window theme. Specific poker client icon might help with localization of these poker windows on the desktop and in the taskbar.

Concerning the table data recognition we will mostly be interested in the following questions: Whose turn is it right now? Is the player that is going to act permanently highlighted or does the observer need to follow the game? Are the card sizes, positions and colors invariable? Is there an information about the total pot? Do we see the size of the blinds, or do we need to catch announcements that blinds are rising? How difficult will the numeric font recognition, card recognition or maybe player name recognition be if necessary?

Table B.1 attached in appendix B.1 is covering all this information for our representative casino set.

2.4.2.3 Game-play related questions

There are also some general game-play questions related to poker robot development for specific servers.

Support of play money tables might help during the initial phases of implementation. We do not need to invest real money to experience the game and to gather all the information from the game that we need.

Table filter might do the job, especially if we want to build a robot that can

automatically join tables and start playing. For tournament play this is an essential feature, although this is important for the ring games as well. The table filter might relieve us of some complications related to correct table searching and joining.

An integrated support for player notes in a casino is probably needless for a robot, it can store and process the information itself. It might however tag a player if it finds out interesting information about him for further human play. Nevertheless, considering the number of active players, it is barely possible to hit somebody with a note that was made before, especially at the lowest stakes. More often, the contribution of making a note is that a player is marked at all tables that are opened and played. Starting multiple games at once is a common practice and the same opponent may be encountered on different tables.

Another very important thing to observe on a casino, is whether it supports the so called turbo sit & go tournaments. Further discussion about this topic will follow in the bot strategy. However, the basic idea is that the sooner the tournament finishes, the less of playing style is revealed and can be observed. This is helpful for a bot in both ways. Observing tactical habits of other players is something that a computer program is not very good at. Likewise, other players might observe patterns in the robot's play and take advantage of them.

Table B.2 attached in appendix B.1 is covering all this information for our representative casino set.

2.4.2.4 Conclusion

As seen from the previous investigation, the approach of graphical recognition is feasible. Although there are some differences among our representatives from the casino set, the design of the recognition subsystem should cover most of the usable casinos. This should ensure that many new and rising casinos following the actual trend will be covered by our environment.

- We will use screen capturing and image recognition in our environment.

Chapter 3

Design

At this point we know enough to start designing the environment for automated online poker player creation. Although we have decided for the graphical recognition approach, we do not want to ignore different approaches. It is possible that for certain poker clients a different form of data gathering will fit better. On the other hand, we need to provide a robust solution for the recognition approach that can handle most of the casinos.

The same ideology applies to the decision making system. We do not want to force the poker algorithm designer to use our approach and related tools as the only possible. He should be free to choose any form of realization for his ideas. On the other hand we should provide an environment powerful enough to make the algorithm implementation easy, fault-resistant and able to cover most of the developer thoughts.

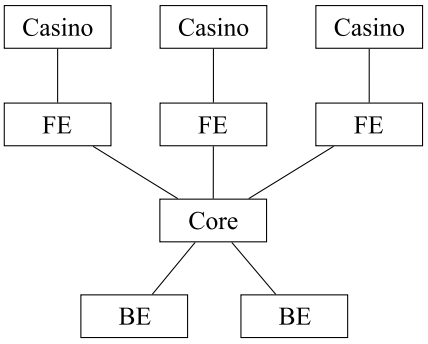


Figure 3.1: Environment component diagram

3.1 Architecture

From the requirements stated above the overall application model depicted at component diagram at Figure 3.1 arises. The front-end is responsible for observation and interaction with specific poker client. It knows nothing about playing poker, its only job is to transform the information provided by the poker client to the specified data structure describing the state of a poker table. In addition, it performs game actions upon requests. These requests are provided for the front-end in specified data structure as well. This is depicted in Figure 3.2.

At the opposite side of the system there is the back-end. Its objective is to transform the representation of a poker table to a specific action that should be executed on given table. The whole process will be mediated by the core. The core may delegate requests to a single back-end, or use an instance of the back-end for each poker table if more complex decisions with preserving information are expected. When more time consuming algorithms are used in the back-end, the core may run them concurrently in different threads to take advantage of todays multi-core systems. Figure 3.2 shows this as well.

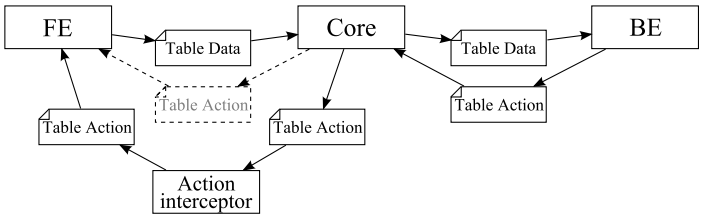


Figure 3.2: Table Data dataflow diagram

Every part of the overall system can be replaced with another one using a different approach if necessary. The only condition that must be retained is the usage of specified data structures for the input and output. We will design these structures later.

3.1.1 Providing additional information

Sometimes the front-end might be capable of extracting additional information regarding a poker table that are not in direct relation with the actual poker table state. Likewise, the back-end might be able to take advantage of such an information and put it into use during the action computation. This might for example be a statistic about players around the table characterizing their likelihood for going all-in, seeing a flop etc. An algorithm knowing such information might take advantage of it when deciding whether to put pressure on somebody or rather fall back.

In this case there must be a mechanism supported by the system design that allows implementing additional information providers and receivers. This can be realized by an abstract parent class for such information. The front-end can generate particular descendants with given information, the back-end will accept the abstract parent and according to the object type it can decide to process it, store it and later use it, or let it go. Figure 3.3 shows a dataflow of optional table data structure.

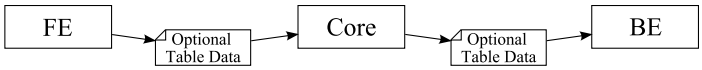


Figure 3.3: Optional Table Data dataflow diagram

3.1.2 Action interception

A request to control and eventually influence the action computed by the back-end could become very handy especially during the algorithm debugging and tuning. To avoid immediate execution of a requested action we create an action interceptor. An object inserted between the core and the front-end that mediate the requested action. Just before the action is passed to the front-end this object can observe, log or even change the action if necessary. The intention is to allow the human operator to approve or even change the requested actions before they are executed. This might be realized through a graphical user interface.

3.1.3 User interface

To control the system, the user interface is a quite important component. A graphical user interface managing the whole functionality of this system seem to be a good choice. It will be responsible for creating the core and a front-end we are willing to use. The back-end is dependent on the game type that is monitored by the front-end. Therefore, the creation of appropriate intelligence (back-end) should be subject to the front-end which knows what is going on. Moreover, the user interface - let us call it a client, must handle the settings of the system, tell the front-end whether to automatically open new tables, which type of tables, display debug logs and overall system condition, or even represent the action interceptor for controlling the actions on all opened tables as stated in the previous subchapter.

3.2 Front-end

The front-end will be specific for each casino. It will be supported only one instance of a front-end running at a time. Theoretically, with a different approach than the

graphical recognition, it might be feasible to run multiple front-ends with multiple casinos at once, but this is definitely not going to work when operating on shared desktop with shared resources such as a keyboard and a mouse.

Although we are not going to force the user to create front-ends in our way, we will propose and later implement an exemplary approach based on screen capturing and emulation of actions on input devices. The following chapter is discussing ideas exactly from this point of view.

3.2.1 Conception

We should beware of parallel resource sharing due to an obvious reason, therefore a sequential approach to every action taken seem reasonable. Let us assume that we keep at our disposal the instruments capable of screen recognition and user input emulation in the way we need. The fundamental decomposition of a front-end might consist of the following parts. A window manager handling the overall window placement, taking care of possible window overlapping and capable of window closing, moving etc. For each window there will be a particular manager handling a specific window type. This is especially the *lobby manager* capable of joining the casino games and the *table manager* whose objective is to monitor a specific poker table. These three parts needs to collaborate very carefully, ideally managed by the window manager. Window manager will call the sub-managers hanging on each monitored window to perform requested actions, ensuring that all the resources are available and can be used by the sub-manager.

When the front-end observes and extracts a relevant information, it will be sent as a table data structure or an additional table data structure to the core, tagged with a unique table identification number. Later the action request identified with that unique number can arrive asynchronously. The front-end must avoid generating the same information based on unchanged table state until the action is executed and the table state is changed. The action request will be passed to the sub-manager at corresponding table to be executed.

3.2.2 Interface

The whole front-end will be encapsulated with an interface for incoming demands. One already mentioned is the interface for accepting requested table actions. Other standard methods that will be provided and implemented by all front-ends follows:

Attach to lobby command will create and initialize the window manager. All windows related to associated poker client will be found, a lobby manager will be created for the poker lobby control. On windows with poker tables a table manager will be created. Other redundant windows that are not used by the front-end

will be closed. At this point the window manager will be responsible for all necessary manipulation with poker windows and no user interaction with the poker client software should be allowed anymore.

Start and *stop* methods on the front-end interface will turn the monitoring of all associated poker windows on and off. When the monitoring is on the front-end will generate (optional) table data structures and send them to the core for further processing.

A mini console for setting front-end specific options will be implemented through a *set configuration* interface with a string parameter. With this mini console it will be possible to universally configure all front-ends from the client.

Method for setting an operational screen area for the front-end is legitimate for all front-ends, not even for those using the screen recognition. This will ensure that the front-end only uses the specific area of the screen, while the rest remains available to the user and other usage.

Setting maximum number of tables that the front-end can start and play simultaneously is the last versatile parameter that is common for all front-ends.

3.2.3 Approach

In this subchapter we will try to propose an approach on how to use all the front-end parts and managers to achieve our goal. Based on this approach we will establish a set of requirements and supportive tools necessary for our front-end implementation.

3.2.3.1 Recognition

We know that a poker table consist of many visual parts combined together to form a complex scenery. Decomposition of this scenery is quite simple. Usually there is a background with chairs around an empty poker table. This is the unchanging base. There are more important elements on this background. For example, there are boxes with player names with fixed positions. Acquiring the information whether a seat is occupied by a player or empty is as simple as recognizing two colors of the appropriate pixel. The pixel color can also be used to locate the dealer button position or to find out whether a player is still in the game with his hand.

Some elements can not be recognized using a single pixel comparison but rather require the information about the whole image. An example of this situation can be provided by the action buttons, which sometimes display “call”, “check” or “muck hand” texts. It could be possible to find the right combination of pixels to do determine the situation, but probably much simpler and user friendlier solution is the image matching and image comparison.

When talking about the action buttons, another really useful feature would be the possibility of pressing such buttons. At first sight this is mostly related to the user input emulation, but deserves a note here as well. Because the graphical recognition part knows the position of the elements that can be clicked, it would be very unsuitable to duplicate this information later in the input emulation. Likewise, providing only a single pixel position for clicking a button may evoke a bot suspicion. Therefore, it would be convenient to specify an action area, which will be used as a parameter for the input emulation. The action area could be used as a single entity, or the images and pixels intended for recognition could declare such area itself according to their position and size. As a result, for example, the fold button can be recognized as an image and then pressed using its area definition deduced from the image parameters.

Concerning not only the poker table, but for example a poker lobby, we figure out that sometimes the yes/no answer for an image presence is not sufficient enough. Simply, sometimes we need to search for things. A requested game to join could be the example. It is displayed somewhere in the client, while we do not know its exact position, but we want to get the position if an object or rather an image is present. A mechanism providing this kind of service should be implemented in our recognition system.

The last and probably the most complicated thing is the optical character recognition. We will mostly need to recognize numbers with additional characters like a dot, stroke, dollar, pound or Euro, sometimes even whole words like "Sitting out", "Raise" or "All in". The cards could be recognized using this method as well. Not only their numbers but also their suits according to the shape of clubs, diamonds, hearts or spades. It seem that a universal character recognition system will be necessary. Moreover, the text positions are not constant. Texts may be centered or shifted to side when a lot of graphics is drawn. On the poker tables there are usually multiple fonts used, sometimes the font color is changing, for example when waiting for a player to turn and the whole player box including his stack is blinking. All of this must be taken into account and supported by our character recognition system.

3.2.3.2 Window localization

We already know what we are going to analyze and recognize on poker tables and we have brought in some requirements for the recognition system. Taking this into consideration, we can try to build the window management on the same approach instead of introducing a different methodology. This might and probably will link the program usage with specific system settings. Such system dependency can confront us with some problems. On the other hand, this is something we need to accept one way or another due to the fact that some data provided by poker clients are located in the windows' title-bars. Moreover, making sure that no unsuitable window

overlapping occurs is only possible when we know the actual window theme. The standard MS Windows 2000 theme will probably do the best job as it is supported on the last three versions of Windows used nowadays. Of course, the approach will be compatible with other Windows themes, but a little bit configuration might be necessary when trying to migrate to another theme.

Let us return to the problem of window localization and positioning. Each poker client uses its own window icon. This might give us a good clue on where to start. These icons are used in the start menu as well. Simply searching for the poker icons in the start menu will give us all the associated poker windows already opened. By right clicking on respective button we get the necessary commands. This way we can easily close or move the window to any location we decide.

With regard to the requirements in the recognition subchapter and a fact that we have not used any additional functionality, it seems as a good choice to implement the window management using the same approach.

3.2.3.3 Input emulation

There is a standard way of emulating mouse and keyboard events. This is all what we need to operate a poker client. The mouse is used for pressing buttons with requested action, keyboard is necessary only for setting a bet of requested size. Although it can be done using a slider and a mouse only, for a robot we will prefer the keyboard. The *user32.dll* offers *mouse_event* and *keybd_event* methods that will sufficiently serve our purposes.

3.3 Recognition system

The recognition system will be used by a front-end designer to recognize specific casino. If we were (and later will be) in the place of the front-end designer, we would expect the recognition system to be as simple as possible. When trying to recognize something, we do not want to spend time providing every parameter concerning our table type, font used, number of seats, or even positions of visual components which are usually specific for each number of seats around a table. A simple query like “is player three present” or “what is the stack of player four” would be the ideal burdensomeness for the front-end designer. The recognition system should reflect the actual table type to answer these queries correctly.

3.3.1 Triggers and recognizers in general

Covering the needs for the recognition system we introduce the concept of recognizers and triggers.

Triggers

Trigger is an object constructed by the front-end to recognize a specific item. For its creation it will only need a unique id, or more precisely, a filename. This object will be capable of answering its build in query every time when inquired by the front-end. There will be triggers of different kinds, each kind for a specific type of recognition. For example, a trigger answering a yes/no question whether player three is present by confronting a specified pixel color, or a trigger returning the stack size of player four. To make things even more simple, it will be possible to group these triggers to lists and query them by index and in loop cycles. The trigger itself is a very simple object, it does not know anything about character recognition or pixel color matching. It is just a box containing an information necessary for the recognition process. For the front-end this is the main and only entry point to the recognition system.

Recognizers

The real drudge is a recognizer object. Although it does not know where, it knows what and how to do and search for. As well as triggers, recognizers are of different kinds are specialized on different types of recognition. The recognizers provide real answers, the triggers just pass them to the front-end. A recognizer can be created using the same way as a trigger, by a unique id, again meant as a filename. Together with a corresponding trigger of the same type they create a recognition apparatus. To get rid of the burden of managing recognizers from the front-end point of view, each trigger is congenitally associated with a particular recognizer through the recognizer id. At this point another object managing these relations can be introduced.

Recognizer manager

The recognizer manager is an object that takes care of the trigger-recognizer association and relieves the front-end from other necessary duties. It also adds the third inevitable piece to recognition process which is the actual screen image used for recognition, provided by the front-end. To correct the original statement about the triggers, they need another parameter besides their unique id to start working. It is the recognizer manager. The front-end supplies the recognizer manager with actual image data, the recognizer manager obtains all recognizers needed by triggers and the triggers have everything they need to get the answer when they are asked. A dependency diagram of related parts of recognition system is depicted at Figure 3.4.

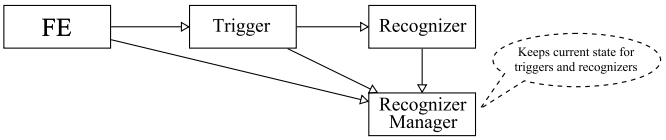


Figure 3.4: Recognition system dependency diagram

Data sharing

We hope that the purpose of separating triggers and recognizers is clear to the reader at this point. It is because of the data sharing. If we want to, for example, determine a position of a dealer button, we only need to know nine or ten possible locations where the button can be drawn and how it looks like. We usually do not need to specify the dealer button appearance for each possible location. Even more eye-striking is the character recognition. The font used for stakes or stacks is the same for each location. Only the position, alignment or sometimes the color may vary. And we do not want to specify the whole font together with each of these changeable attributes. Figure 3.5 shows an example on possible trigger and recognizer structure.

In general, triggers are objects holding the variable parameters, while the recognizers are objects holding the invariable ones. It is obvious and also intended that many triggers will use the same recognizer, together creating unique queries for every visual component that is going to be recognized.

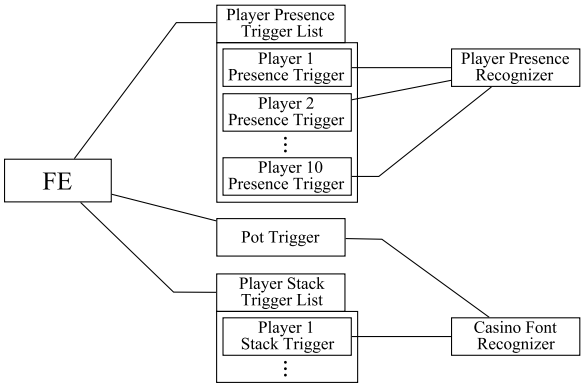


Figure 3.5: Example of triggers and recognizers

3.3.2 Particular triggers and recognizers

The individual triggers and recognizers are based on the exact needs claimed earlier and will cover all needs of the recognition system. Let us take a look at them separately.

Pixel recognizer

The pixel recognizer is the most simple of all recognizers. Its job is to distinguish whether a pixel is of given color with given tolerance for each RGB color component. These two values (pixel color and tolerance) are the pixel recognizer data.

Pixel trigger

Again, the simplest of all triggers. It knows the position of a pixel which is to be recognized. Also contains the filename of related pixel recognizer. Can be grouped into a pixel trigger list and queried by index.

Image recognizer

The image recognizer is responsible for image comparison. It contains a referential image and a color tolerance for each RGB channel.

Image trigger

The image trigger is similar to the pixel trigger. It contains a position where the recognized image should be located, given by the upper left coordinate. It is linked to an image recognizer through the related recognizer filename. Just as the pixel recognizer can be grouped into an image trigger list with indexed access.

Image finder

Image finder is a special type of image trigger. Unlike the image trigger, it contains an area for image searching instead of a single coordinate. The result is different as well. Instead of a yes/no answer it returns a list of all upper-left coordinates of areas where the appropriate image was located. Everything else remains the same as in image trigger.

Font recognizer

This one is understandably the most complicated. It contains a set of all characters that are to be recognized. The color and color tolerance of the recognized font is necessary as well. In addition to other recognizers, it has an option to invert the color matching process. This allows to match the background color instead of the foreground font color. As a result, recognition of different font colors on a given background is possible. To fulfill our specific needs for recognition, each character has a termination flag. After matching that kind of character the recognition

is stopped. This is useful especially when recognizing whole words like “Disconnected” or “All in”. Instead of identifying every letter of these words, or the whole word as a single character, we could only identify the first character. If it is unique the termination flag is set.

Text trigger

Because of its association with related font recognizer, the text trigger is an entry point to the text recognition. It contains locations where the font recognizer is to be applied given by the coordinate of upper left point. Another important attributes are whether to search for the text to the left or to the right, maximal distance to search for a text and maximal tolerated space between two characters of recognized text. Combination of the text trigger and font recognizer covers almost everything we demanded. Almost everything, but not all what we required. At this point, the font recognition only works with given font color or with given background color. In some cases we might face the same font used in both ways, with solid font color or solid background color. To avoid creating another font recognizer with the same letter shapes but different colors, an option to override font colors is added to the text trigger. With this feature it is possible to recognize texts with different text colors or text backgrounds using the same text recognizer. But this is really a special case, therefore all parameters regarding the font color are placed in the recognizer. It is not necessary to specify the font color in every text trigger, as in most cases it is the same for all texts using the given font.

Area definition

Area definition falls under the trigger category. It is the only trigger that has no recognizer counterpart. It defines an area by its upper-left and bottom-right coordinates. It does not provide any return values either. The area definition is only used for user input purposes. Triggers like pixel trigger and image trigger can be also used as an area definition.

3.3.3 Trigger editor

At this point we know what kind of information each trigger and recognizer needs. Obtaining these information manually is certainly possible, but very uncomfortable. We have to build a trigger editor that makes this process as simple as possible. The fundamental principle will be a graphical WYSIWYG editor, where every trigger or recognizer can be created with just a few clicks.

This editor allows us to open a saved picture containing a poker client window or even take an actual screen shot from our desktop with running poker software.

When creating a trigger that contains specific position of the recognized window, it is necessary to make sure the screen shot is taken from the correct position. The editor takes a picture of desktop behind itself with the zero coordinate located in upper-left corner. Subsequently, it is possible to lock the screen shot position and move the editor away to see what is actually happening on the table which is being recognized.

After a screen shot is created in the editor, we can start making triggers and recognizers. We select the recognizer/trigger mode for target recognizer/trigger type, fill in its path and filename and we are ready. All associated values and data fields for the selected recognizer/trigger type are displayed and can be changed directly, or more comfortably in the editor image area by selecting requested areas and positions using the mouse. The tool tip help is present for each data type and editor control. When everything is set, we just click the create button and our recognizer/trigger is ready to be used in the front-end. It is possible to load existing recognizers/triggers to the editor later to adjust their settings.

These were general information about using the trigger editor. There is much more functionality specific for each recognizer or a group of triggers. Let us intimate the reader with it. We will only pick the most interesting parts and functions. We do not want to bore the reader with self-explaining functions and features that can be understood at the first sight or with the try and see approach.

Font recognizer

Compared to the pixel and image recognizers featuring only necessary data values, the font recognizer mode supports a tool used for building font recognition characteristic. In the trigger editor it is possible to add each character that will be recognized by actual font recognizer and provide its string representation used in recognition output when the character is matched. It is not possible to load the font recognizer in the standard way described before. To load a user intelligible representation of the font characteristic, the load button in the font building tool must be used instead.

Triggers

For all triggers an additional feature set is available. It is located on the right side of the editor. By pressing the main load button it is possible to run the actual trigger on the current image and see instantly whether it works correctly. Both the trigger we want to run and its associated recognizer must be already created. It is possible to add multiple triggers to the loaded trigger list and test them on several images just by pressing the run button. When a fix is made on the associated recognizer it is necessary to press the “reload recognizers” button, otherwise the old recognizer

is used further. As we mentioned before, it is possible to group triggers of the same kind to a trigger list. By pressing the "save list" button all loaded triggers are used to create the resulting trigger list. To edit this list later, it is possible to load it, change it, and save it again.

Useful tips and use cases

These tips are intended for someone who is going to use the editor in practice for the first time. However, the knowledge of the trigger editor usage can be extended for an inactive user as well.

The trigger editor starts exactly in the upper left corner of the screen. If the poker client or the front-end uses this location as the standard location for window positioning it is possible to take an advantage of it. By marking the lock screen position in the editor after it starts we don't have to care about placement of windows we want to recognize.

Each recognizer and trigger has its own folder with a unique name. These folders are usually located in the parent folders named "Recognizers" and "Triggers". It is not necessary to provide the full path to these parent folders when operating (creating, loading) recognizers and triggers in the editor. The editor automatically adjust the path if necessary. This is very handy when creating recognizer-trigger pairs, as we are not forced to modify the path every time.

It is not necessary to fill in the associated recognizer's path when creating a trigger. If the associated recognizer has the same name as the trigger and if it is located in the appropriate folder as described above, the editor fills it in automatically during trigger creation.

We can load a created trigger to the trigger list, then modify and replace the original trigger without loosing the loaded one.

When we intend to create a trigger list, we can easily create a list of triggers using the same trigger name by loading and replacing the old one. After all, by pressing the save list button a list of triggers will replace the original temporary name used only for single trigger creation and loading.

The recognizer and trigger data are stored in a file text file and eventually in a graphical format. They copy the appropriate data structure of the given trigger or recognizer type. More information regarding the data structure can be found in program documentation[1], chapter 3.3.4.4 provides more information about the data types. Sometimes it might be easier to copy, paste and modify the existing files of a trigger or recognizer than to use the editor. This is especially true in cases when we need a similar type of trigger, using only a slightly different settings and when it is more time consuming to wait for the casino client do display the new values

[1] More information about program documentation can be found in appendix A and appendix A.1.

which we want to recognize. Creating a dealer button might be an example of this situation. Let us assume that we have already created a pixel trigger list recognizing whether a player is present at the table. This was quite simple, it was only necessary to take a picture with a full table and then use the click, create and load approach to generate the trigger list. Now, for example, if the dealer is marked with a button whose relative position to our pixel indicating player presence is the same, it is easier to modify the text representation of these triggers instead of waiting a whole round to capture all possible dealer positions using the editor.

It is important to be careful when creating a character set in the font recognizer. The characters must adhere to the bottom line. This means we need to include some additional space above the characters which are not as high as the highest character of the whole set. We should preferably use the same height for all characters.

Sometimes, conflicts in the character sets may occur. This happens when one character is a sub-part of another character. In this case the first match is used according to the character order in the list. Matching an incorrect sub-character on another character may result into a recognition failure.

When creating an image recognizer, the editor requires a referential pixel of respective image. This is a simple optimization when a faster pixel recognition may substitute the slower image recognition. If possible, we should try to choose a pixel that is unique for the image.

3.3.4 Implementation

There are many libraries on the market that support optical character recognition and two dimensional image matching. Most of them are paid for and provide many additional functions and features that we will never use or that could make their use difficult in our conditions. We do not want to use approximations during recognition, curved text matching, image rotation analysis or any other advanced feature. We just need a simple answer whether two images match or whether an object is letter A, B or nothing. After some time spent searching for applicable libraries it shows up that it would be easier to implement all these functions according to our needs. Moreover, this way we are not limited by the licences of these libraries.

Although there are many super fast, intelligent and universal algorithms for optical character recognition and pattern matching, we are not interested in these very much in our work. The simplest and most straightforward solution that can do the job is enough for us at this point. We do not want to take every method and approach we have used into pieces. We will just pick some highlights and interesting parts of the recognition process we have implemented.

The module *Recognizers* contain library implementations of all mentioned triggers, recognizers and eventually the trigger lists. The module *TriggerEditor* implements an executable trigger editor with the functionality described before.

3.3.4.1 Image access

There are standard built-in types for image handling, such as the Bitmap type in .NET[2]. Although this type provides the GetPixel method, which is actually everything we need to implement our recognizers, it is very slow due to the access protection and locking. An easy workaround to speed thing up was necessary. This involves the following:

- Encapsulation of the bitmap with another object providing the same GetPixel method.
- Using the access lock and unlock only at bitmap creation and deletion instead of locking and unlocking on every single GetPixel call.
- Allowing the unsafe code in .NET library to enable direct memory access at requested pixel position in the bitmap with a type-cast to create the return value.

This has significantly increased the speed of image processing.

3.3.4.2 Image matching

Two dimensional image matching was done by the simplest possible approach, pixel by pixel comparison. An enhancement such as definition of a key pixel on the bitmap speeds even the whole screen image matching up to an acceptable time for every reasonable bitmap. As it turned out later, is was not necessary to use this approach on the whole screen but only on small sub-areas.

On different computer stations the same images may look differently, meaning we need to proceed carefully. Icons in the start menu are an example. Some color tolerance for the image matching is therefore necessary. This problem might be caused by different monitor profiles combined with different graphical drivers or by the CRT vs LCD issue. We have not investigated this in detail, as there was an easy and quick workaround to solve it.

3.3.4.3 Optical character recognition

There is quite enough character recognition necessary on a poker table. Therefore, the chracter recognition was optimized for our environment. It should work and be fast enough even for image matching of each character when using the pixel by pixel comparison.

[2]We have chosen c# and .NET for implementation of our environment because of the requirements of the back-end. We will discuss them in section related to the back-end.

Each font recognizer builds a map of hash codes unique for every character. This map is based on a few pixels explicitly defining each character. These pixels are generated automatically according to all characters used in the font on recognizer creation. This ensures that there is no need to operate with original image representation of each character during font recognition. It is also not necessary to match the whole area of recognized character with each font character.

Unfortunately, this approach bears a possible conflict between a character and a sequence of characters that can be matched vice versa under some specific conditions. Most of these conflicts are automatically solved during the font recognizer creation, however the user must keep in mind that the correct order of characters is necessary in case of unresolvable conflicts. In short, we should move the wider characters up in the list when a recognition conflict occurs by accident. This was not done automatically due to better lucidity and rare occurrence of unresolvable conflicts. For example this might happen when using a dot without extra space on its sides and a low line. The recognition system can not distinguish between a low line and four or five dots next to each other. It uses a greedy approach and the first match in the font character list.

3.3.4.4 Serialization

To store all the data needed by recognizers and triggers in a user friendly and editable text form, a module called *MySerialization* was created. It provides serialization and deserialization of basic types such as boolean, integer, string, color, coordinate, bitmap. MySerialization module is used by appropriate recognizer and trigger data structures holding their internal data to serialize and deserialize recognizers and triggers.

Graphical data are stored in PNG format, other data types are stored in text format sequentially. Each data entry is stored on a new line in brackets. The values of boolean type are *TRUE* and *FALSE*, color is stored as a sequence of three color components, red, green and blue, delimited with a comma in range from 0 to 255 each. A coordinate is stored as x and y components, again delimited with a comma.

3.4 The core

The core is in fact a very simple meeting point for all involved parts. A lot of core functionality has already been described before. Just for recapitulation, the core allows the front-end to register a new poker table with a given back-end and algorithm. A unique table id is used. Afterwards the front-end sends table data and eventually optional table data to the core for processing. A detailed view on table data flow is depicted in Figure 3.2, Figure 3.3 shows the optional table data flow. A

separate process is created for every algorithm call. The algorithms run in parallel, taking advantage of multi-core processors in case of time consuming computations. The core does not know anything about back-ends. It just needs to send the right table data to the right back-end and then receive the action request and associate it with the right table again. Now, there are two ways how to relegate the requested table action. Send it back directly to the front-end to be immediately executed, or if possible, send it to an object implementing the *IActionInterceptor* interface. Bringing back to memory, it is usually the client that does the logging and allows action modifications to the user. This is everything that the core must handle.

The idea of multiple threads for every table action brings another significant advantage. It is very simple to implement sequential action processing in the client and in the front-end. The front-end may process the whole screen, sending as many information as possible. On the other hand it will pick up the results one by one and process them when idle and the resources are available. Compared to a single-threaded model, when just one kind of table data is created, processed in the back-end and returned back to the front-end which requires implementation of additional techniques to avoid other table starvation or implementation of queues with outgoing and incoming data. The same applies not only to the front-end but to the action interceptor – the client - as well. It may or may not support processing of multiple action requests at once. It can display a user dialogue and after it is answered display another one, taking advantage of the queue of waiting threads or just simply display multiple dialogues for every action, so the user can decide the order of processing according to the importance or whatever he wants.

Just a note to the implementation, the core is named “thread manager” after its main purpose and is implemented in the the same name module called the *Thread-Manager*.

3.5 Back-end

As said may times earlier, the most important task of the back-end is to transform table data structure describing the actual table state to requested action for given table. The back-end also needs to accept additional table data which provide supplementary information gathered from the poker table. It may or may not take such information into account and make the best of it. What was said about the front-end, that we do not want to force user to any specific approach, applies to the back-end even more. The variety of possible implementations is uncountable. Some of them are easy to use, some of them cover wider areas and give the algorithm programmer more power. To make our environment fully and relatively easily usable, we have implemented one new exemplary approach that goes together mostly with the no-limit tournaments. While other parts of the system like the front-end and the

client are intended for the programmers, this back-end part must be open even to less skilled users. This means that no advanced programming technique should be required.

3.5.1 AI

Our idea on how to program a poker robot intelligence has been elaborated long before any factual investigation of other concurrent software. Even after the research our approach seems to be the best and most suitable for no-limit tournament implementation. This wouldn't be true if there was a winning algorithm that could be adjusted and be profitable by sliders in a few minutes.

We have no huge poker hand databases from the real events at our disposal, therefore any approach based on statistical analysis and overall results in similar situations in not feasible. We know that we want to make a rule-based system. Rules like "have aces?", "Go all in!" look good and are often the base of other programs, but do not cover enough situations to make a good decision. Sometimes actions depend more on other factors than the cards. It might be useful to calculate other values before making a decision. The complication of these computations must not be limited by the environment. Therefore, we want to use a full-value programming language if necessary. Building that kind of a language with a lot of poker built-in functions and types might seem satisfactory. On the other hand, how many additional functions and standard algorithms that may come handy are implemented in such a language? The best solution seems to be to use an existing language and extend it with additional data types and functions related to Texas hold 'em poker. With a template or a sample implementation of a back-end algorithm, it will provide an easy and at the same time a very strong tool for poker intelligence implementation. We have chosen c# and .NET as it is a modern and very user-friendly programming language fulfilling our needs for the hosting language for our environment.

At this point there is still nothing new compared to other approaches. For example the *Meerkat-API*, used by the *Poker Academy Pro* software developed by the games group of University of Alberta for implementing robots capable of playing in their environment uses the same approach. A class derived from the base class *Player* and especially the *getAction* method can be implemented in java language and compiled against the provided meerkat-api library. After copying the compiled .jar file to the Poker Academy bots directory, it is possible to load the new bot and see how it does against other already implemented bots. A good way of starting poker robot implementation might be to use this api, figure out whether we are capable of creating an intelligence that can compete with their existing robots and just after this try to emulate this interface on a real casino. The biggest drawback of this approach is the need to recompile and restart the robot after each change. During

the creation and debugging of the algorithm there are many changes needed. Another drawback is that this software suffers by the fixed-limit ring game approach of the authors, although it is probably the best on the market nowadays. A short summarizing review of the Poker Academy Pro software from the point of view of sit and go tournaments can be found at PartTime Poker[3].

We will use the same approach that allows the user to create a dynamic library with the algorithm in a standard programming language and environment. For this purpose the .NET and especially the c# in combination with the Microsoft Visual Studio seem to be a very good choice. It provides a user friendly runtime debugging with code change support during application runtime. This is something that will help the algorithm developers and allow them to do step by step observations of their algorithm. In case that something is wrong, they can correct it real time and continue with the same situation without the need to restart and model the exact situation again.

At this point we are proud to present a new idea on how to program the poker robots, especially for the no-limit tournaments. In ring games, especially in the fixed-limit version, it is easy to compute probabilities of making the best hand, or having the hand remain the best to the end. Adjusting bets according to these probabilities and giving the opponents wrong odds is the key to make profit. If the situation appears unprofitable, a player simply folds and waits for a better hand. We do not imply that there is no bluffing in the ring games. There indeed is, at least in the no-limit version. But it is really rare compared to the tournaments. In tournament games it is not only about the current situation on a table. Here we usually do not have the luxury to afford folding and waiting for a better hand. As the blinds are rising, more and more bluffing occurs in a desperate attack on the pot. We used the term bluff, although it might be the correct decision. It was used just for emphasize that in tournaments we need to make actions that are considered as extremely bad play in the ring games. Instead of focusing on the exact game situation, here we need to see every table state as a part of a sequence beginning with dealing a hand. We need to observe betting patterns and take advantage of these patterns. This is something that is crucial and also hard to catch in the algorithm programming environment.

- Instead of having a single entry point for the action computation in the algorithm we will build a state automaton, where every state matches a specific betting pattern and relevant information from the previous rounds.

This allows the algorithm programmer to keep track with what happened earlier in previous rounds in a suitable form and granularity. It is up to him to choose number of transitions leading out from each state to cover the information he will need later.

[3]See the article at *http://www.parttimepoker.com/review-of-poker-academy-pro-2.*

An example on why is this approach an ultimate method in the no-limit tournament programming follows.

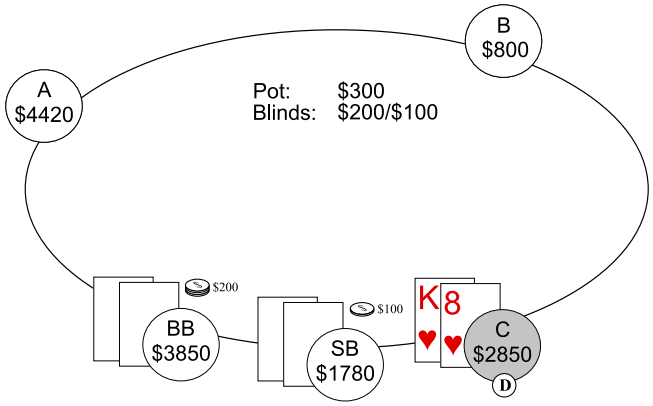


Figure 3.6: Blind steal example

Let's imagine a single table on-line tournament with three places paid in a middle phase with blinds $200/$100. Players A and B folds, we are on the button with K♥8♥. A decent hand to try to steal the pot. This situations is depicted in Figure 3.6. Our algorithm knows that too, so it raises to $500 in hope to get the pot without any action. Small blind folds, but the big blind calls for the remaining $300. The fop comes A♠Q♣3♥ and the big blind checks. What should the bot do now?

In case of the original approach with a single entry point we have two options. We can decide for an action based only on current table situation. This means that we have nothing, therefore we check too. We can also try to make a decomposition of the actual situation. The pot with $1100 in it compared to the actual blinds is indicating that some betting already occurred. Based on our position on the button we might deduce that we have been stealing pre-flop. Unfortunately, there are more possible scenarios which make sense and lead to the exact situation we are facing now. One of players A or B might have limped in, but has folded to a raise to $400 made by player C or the big blind. As we are player C, it would be very good to know who has the betting lead to solve the current situation correctly. The point is, that the decomposition process might be ambiguous, unsatisfactory and also very difficult to make.

How will this situation be handled with our state automation approach? We know we were stealing on pre-flop so after the flop we found ourselves in the *FLOP_button_steal_failed_and_called* state. All other situations with different betting patterns will result in another state or our fold. What should we do now? We know that when we are in this state we have shown a pre-flop strength. We know that our opponent did not show any strength after the flop as he only checked. On

the table we see a scary ace card and the queen of which the opponent might also be afraid of. We might take advantage of this situation and bet again to make him think we were raising the pre-flop with an ace-something, or a connected high card combination such as queen-king, or jack-queen.

To make this poker example complete, we will finish the betting ideas although they are a little bit off topic at this place. We do not want to waste a significant part of our stack for the bet. If the opponent flopped a good hand and is slow-playing it, than he will call everything and we are probably going to be beaten. A smaller bet might do a better job. This way we will tell him we want to be called. What is more important for us, if we face resistance, we are not risking that much compared to the possible revenue. A bet of $400 - $500 would be accurate. In our state automaton this means, we want to raise to 1/3 of the pot which is rounded to $400 and in case we get called or re-raised go to the *TURN_RIVER_beaten_miracle* state. Otherwise we will collect the pot and the hand is over in our favor. In that new state we do not want to spend any chips, unless a miracle like two other hearts or a ten with a jack appears to give us the king high flush or the best possible straight.

In fact our opponent folded his T♦J♦ probably thinking that only the king makes his winning hand. With a chance of 47:4 compared to the 15:4 odds offered by the pot it was a good play[4].

3.5.2 State automaton implementation

The state automaton for poker algorithm representation will be implemented via methods in the user algorithm class. Every method will represent one algorithm state. Of course, any state may call other states for solving a specific sub-problem. There are following requirements on the states:

- Every state must return an action that will be executed.
- It must be possible to locate the algorithm state which computed the resulting action in the source code and tell it to the algorithm programmer.
- Every non-terminal action request must define state successors for the next action computation.

The first condition can be ensured through a return value of every state in compile time. The second can be supplied by using the exceptions as the return value for requested state. These exceptions contains the source file and the position in the source file where the exception was raised. This way we are able to identify the appropriate state that raised the exception, moreover the exact clause in the algorithm.

[4]There are four kings among the remaining 47 cards, he does not know we have one. And if he wanted to continue playing for the pot with $1500, he must put in additional $400.

Although these exceptions may be hidden to the algorithm programmer in an object constructor used to create action request, we will prefer the direct exception rising in the algorithm state due to a better syntax highlighting and lucidity. The third condition is also easy to fulfill. Every action that requires definition of successor states accept the references to the corresponding states as parameters. In c# this can be done using the delegate construct.

3.5.3 Algorithm implementation

Every back-end implements one algorithm. It must implement the *IAlgorithm* interface as a standard entry point to the back-end and its algorithm. At the end of one computation it returns the requested action through a call-back method to the core. What is behind this interface is completely up to the algorithm programmer. With the state automaton approach an *AlgorithmHelper* module is providing all the necessary background. It requires two basic automaton states, a *preflop* state, and a *backup* state. The preflop state is called when a new hand is dealt and no other action occurred with this hand. The backup state is called when something went wrong during the automated play, but especially in cases when the algorithm was started with a hand already in progress and the betting pattern is missing until this point. After receiving a turn request with actual table data in the back-end it is only necessary to delegate this request to the AlgorithmHelper through the StartAlgorithm method. Everything else is done for the user when adhering the algorithm programming rules.

3.5.3.1 Automaton states

Two obligatory states necessary for using the algorithm helper were already described. Creation of other states is completely up to the user. It is possible to use more source files for each specific set of states. The preflop and the backup states are implemented the same way. A standard source file implementing a concrete algorithm may look like as follows:

```
using NS_AlgorithmHelper;
using NS_HoldemPoker;
namespace NS_MyAlgorithmNamespace
{
    internal partial class Algorithm
    {
        internal State MyState1()
        {
            // The state code goes here...
        }
    }
}
```

There are namespaces providing the basic hold 'em poker terminology and functions, which we are going to talk about later. Here we provide an example of two that will probably be used more. It is also possible to include and use other third party libraries or .NET built-in functions and algorithms.

3.5.3.2 Automaton action

Generally, two types of actions are available. Terminal actions that finish the current hand right away without any further decision making and actions that change the state of the automaton for further decision making. *Fold* and *all-in* actions belong to the first category. After making one of these moves the algorithm starts again in the prefolp state with a new hand. Of course, only if we are still in the game. Actions from the second category are: *check-fold*, *call* and *raise*. For these actions it is necessary to define the next states of the automaton. Each of these actions requires two states, one for repeated action in the same betting round and one for the next betting round. If someone raises or re-raises after our move we usually want to continue in a different state than if no significant action behind was taken and a new card or cards were dealt. An example on these two action types follows.

```
If (tdata.myhand.GetCard(0).n == ECardVal.nA) // if we have an ace
    throw new ActionCall(NextTurnState, RepeatedTurnState);
else
    throw new ActionFold();
```

This is what our algorithm's implementation is about. Creating states and distinguishing the appropriate actions together with defining the next states make a very strong instrument for creating no-limit tournament algorithms.

3.5.4 Poker data types and functions

To make the automaton state implementation efficient and well-arranged, a lot of additional poker functions and data types were implemented for the algorithm designer. All of these structures and functions are located in the *HoldemPoker* module. We will only take a general look at them, a complete list of these functions and types can be found in the program documentation under appropriate module and namespace.

3.5.4.1 Table data

The *TableData* structure provides all information regarding the actual table state. It holds information about blinds, player stacks and bets, size of the pot, our cards etc. Everything that needs to be said to recreate the situation on the table. It is passed to the back-end through the *Turn* method every time the turn is called. This is probably

the most important and used structure in the algorithm. This structure is defined in the *NS_HoldemPoker.TableData* namespace.

3.5.4.2 Supportive types

Hand, *card*, *player state* are examples of other types used by the table data structure. They allow us to easily operate with the same name terms. We can ask for a card in our hand, ask for its value or color, compare it to another card or another card value or color etc. These supportive types define whether a player is active or whether was disconnected and many more. All this supportive types are located in the *NS_HoldemPoker* namespace.

An important thing to know is that the cards in our hand are sorted. We can be sure that the first card number is at least of the same value as the second card number in our hand. This is because the order of cards is irrelevant and this makes it easier to implement some conditions and queries with our hand.

3.5.4.3 Helper functions

This is another category of functions that spares us a lot of time during the algorithm implementation. All of these functions need the actual table data passed as a parameter to compute their output. They return very useful information regarding for example number of players acting after us, number of callers, raisers, all-in players in front of us, computes the maximal bet or maximal possible re-raise we may face later and answers questions whether a card of given number is on the table, or how many suited cards are on the table. To use these functions we need to include the *NS_HoldemPoker.HoldemPokerQuery* namespace in our algorithm file.

3.5.4.4 Hand queries

To find out how we stand with our actual hand and given situation on the table we can and probably will use the "is hand query" functions. The same namespace and the same TableData parameter is required, exactly as in the helper functions mentioned before. With these hand queries we can ask whether we have two pairs, a set, or just the middle pair and so on. We can also ask whether we are drawing to a flush or to a straight as well as whether we made one already.

3.5.4.5 Internal functions

We know that it is not possible to cover every possible hand strength with these functions. Sometimes even the very definition of the hand can be questionable. For example, the straight draw. There is a significant difference between an open and inside straight draw. Are we interested in a set when all the three cards are on

the table? Some of the functions can be implemented directly and locally by the algorithm creator according to his or her specific needs. For this purpose, a small set of internal helper functions was created that may help him with implementation of his specific helper functions. He can ask for example for the next occupied seat position from given position, or to sort an array of cards according to the card numbers. The namespace it is necessary to use with this set of functions is the *NS_HoldemPoker.Internal*.

3.5.5 Programming summary

Everything fundamental related to the algorithm programing has been revealed. The combination of multiple files containing different algorithm states and many supportive functions gives us a very transparent and effective tool for poker algorithm creation. This can be boosted up with the Microsoft Visual Studio and the runtime algorithm modification. The information including the filename and the exact location where our action request occurred in the algorithm makes the debugging and algorithm bug solving as simple as it can be compared to the strength we keep at disposal.

3.6 Testing

Testing and especially the regression testing of a system is often shifted aside in many products. Everyone knows that it needs to be done, but nobody actually does it thoroughly. In our case this is something we really can not underestimate. The algorithm will probably be adjusted, improved and tuned all the time. Small ideas on improvement that solve a specific situation will be added during the lifetime of the algorithm, usually inspired by specific situation and the algorithm game play that we saw. Adding even the most fundamental improvements may sometimes result in a creation of another small and hidden problem in the algorithm. Bigger changes usually result into automaton state-flow modifications. These types of bugs are usually disclosed sooner or later during algorithm play, but it would be better to find them right after the corresponding change that caused them, to keep the track with an overall idea of the algorithm. Some errors may occur even in the early phases of implementation of the recognition phase. It would be proper to test the correctness of the recognition as well and watch for of accidental “fixes” that may spoil the recognition process.

To do so, we will implement a complex and integrated system for regression testing. It is not possible to cover every situation that can happen in hold ’em poker. A much better approach would be to create the test set gradually, based on real

situations that we observed during game play and that were interesting, or played incorrectly by the algorithm.

- It should be possible to recreate the tested situation for the algorithm (and eventually for the front-end as well) to see whether a fix or current version of the algorithm acts correctly according to given situation.

At first sight this might seem as a trivial problem of just resending the actual table data to the algorithm. Or more likely, to resend all the data beginning at dealing the actual hand to the current situation we want to test, in order to induce the betting pattern for the algorithm. On the other hand, this might still not be enough. There are two general problems. The more substantial is the possibility that the algorithm has built some kind of player profiles according to additional observations provided by the optional table data from the front-end. These observations might be time dependent, for example monitoring the frequency of blind steals of each player, or database dependent, when a player profile and profitability can be used in algorithm decision making process. Of course, the database can be changed during the time. Therefore, even the same situation of playing one hand can theoretically result in a different action. This problem is mostly ideological and we are definitely not going to face such problems in the early and middle phases of the algorithm implementation. The other problem interferes with our intentions more, it is the necessity of using random decision making in the algorithm. Simply, the algorithm cannot play similar situations in the same way and sometimes needs to randomize its decisions. This is the problem we are more likely going to face in the regression testing.

When we face a situation that we want to use for algorithm adjustment or for later regression testing, we want to be able to create a sequence from the dealing of actual hand to the current situation, in best case to the end of actual hand. Moreover, we want to preserve the same "random" decision making in the algorithm. If we allow the algorithm to make another decision somewhere before the critical moment, we may not reach that critical moment at all. Even more, we cannot continue the testing, because actions of other players are conditional on our move. We simply do not have the right data available for the later betting rounds. Ensuring this requires a random decision making management connected with the regression testing sub-system to make the process hidden to the algorithm.

The other important decision that needs to be made is to choose the data type for later testing. The simplest solution would be to store the table data structures for every hand. The simplest for us, but very uncomfortable for the tester. It would be very difficult and time consuming to realize what is going on only according to a text representation of poker table state. It would be necessary to make a tool that reads the data and creates a visual representation of the table state that could be understood in very short time. This approach also avoids the front-end testing completely. Therefore, we will prefer the exact graphical representation of the table

from the casino clients and simulate the adequate casino client window. The front-end will not know, whether it is playing in a real casino or in our simulated window. However, the client must know what is going on. To make the testing faster, it will not register for the action intercepting through the *IActionInterceptor* interface. The random number generator must work in a special mode, to make the algorithm do exactly the same "random" decision that were made when the test case was created.

It is recommended to use the *RandomNumberGenerator* module to get random numbers or random decisions. Especially in the back-end implementation we should use the random sequence generator and its method *IsPercentage* that returns *true* or *false* according to the percentage parameter and allows us to make easy random splits in actions. Not only will we avoid the testing problems this way, we also get an unpredictable sequence of decisions randomized automatically during a game.

3.6.1 Test scenario

Test scenario will be represented by a series of images of a poker table capturing crucial moments of one hand. These moments are the table states just before our action has been taken and the same pictures that were used in the front-end table recognition process. These images will be supplemented by a short text script describing the situation and actions that should be taken by the algorithm. To give a standard form for these text scenarios, let us say the images will be named with numbers from zero in order they were created and will be stored in a no-loss compression format PNG. The script describing the test sequence order and expected actions will be stored in a "test.txt" file. These images and the file with script located in a single folder create a test scenario.

Test scenario commands

To be able to correctly replay the sequence of images in a scenario and confront the real action taken with the expected action, a few commands that cover these cases are necessary. The complete set of commands includes: *DISPLAY*, *FOLD*, *CALL*, *RAISE*, *ALLIN*, *END*, these are modified by the user, followed by *RANDOM* and *FAIL* commands used by the scenario creation mechanism. A scenario may look like as follows:

```
DISPLAY 0.PNG
CALL
DISPLAY 1.PNG
RAISE 120
DISPLAY 2.PNG
FOLD
END
```

The DISPLAY command takes a parameter with image filename that should be displayed and is followed by one of the four commands representing the action that should be taken. The RAISE command takes a parameter with the amount of chips to be raised. After reaching the END command the scenario is marked as correctly passed.

When a new test scenario is created using the module *TestScenarioCreator* and the standard approach, the scenario script contains a full sequence of captured images to be displayed, each followed by the FAIL command. After reaching this command the scenario is immediately finished and marked as failed. The user must fill in the correct betting sequences before the testing. Another possibility would be to do it after the testing, according to a log with failed scenarios. Sometimes the test scenario script contains the RANDOM command at the beginning followed by its parameters. This command is used to regulate the random decision making in our algorithm to achieve exactly the same "random" decisions that were made when the test scenario was created.

During the implementation and testing of our algorithm we found out that the newly created and captured scenarios should not be included in the regression testing as they usually fail. We have therefore added a *SKIP* command which should be used at the beginning of a script. This ensures that this scenario will not be marked as failed in the resulting summary. Right after an appropriate fix is done, it can be removed and an actual test case will extend the test suite.

3.6.2 Tester application

To be able to test these test scenarios in our poker playing software we will need an application that is capable of emulating the poker situations according to the test scenarios. This application has to work in different modes, depending on the casino type and table structure where the test scenario was created. The objective of this application is to traverse through the directories containing captured test scenarios and replay them. According to the commands in attached text script it will emulate requested table situation and wait for actions to be taken. If the action is correct it continues to the next command until the final END command is reached and scenario is marked as "OK". When a wrong action is taken then it interrupts the scenario, creates a log and continues with another testcase. After processing all scenarios it creates a summary log with number of failed, passed and skipped test scenarios. User can afterwards search through the log to find those that failed.

3.6.3 Test scenario creation

The module *TestScenarioCreator* implements everything necessary for easy test scenario creation. The rest of the burden is on the front-end, however the client

needs to participate on this process as well. The front-end supplies the test creator with all necessary data, the client tells the creator which test scenario should be created and which not, mostly according to the user requests. Test scenario creator accepts an id of a scenario, actual screen-shot of appropriate window and a position of relevant data in the window. This is provided by the front-end every time when a player move is detected and table analysis is started. When the hand is finished, or rather a new hand is dealt on appropriate table, the front-end tells the scenario creator to close a scenario of given id. That is all that the front-end needs to do. But only this will discard the test scenario and prepares the creator for another one under the same id. Which by the way means that it is possible to use the table id to create test scenarios using the scenario creator. To create the real output it is necessary to tell the creator we are interested in providing the appropriate scenario id. When a test scenario with a positive create request is closed, at first it is created, saved and just afterwards discarded. This ensures that it does not matter when we decide to create a scenario based on the actual hand. The scenario covers everything from the hand dealing to the end, it does not matter that we have decided to create it, for example, before the third betting round.

If we are only interested in a situation that happened early and do not want to test to the rest of the scenario to the end, we simply delete all redundant images and adjust the test scenario script.

Let us return back to the problem mentioned at the beginning of this chapter, concerning the player profiles that were created in the algorithm during the whole game-play and how to evoke the exact state of the algorithm when the scenario was taken. We have sidetracked it on purpose, to have an idea of how the testing process works. We can not provide any universal solutions, or at least we doubt about usability of such solution. However, the user and programmer of the system has a way how to deal with it. Providing any algorithm specific data can be done using the same approach as we have used with the random decision making. We do not know what type and form of data the user may need to store and use with the algorithm before each tested hand, but he may extend the test scenario creator to accept more information in any type he needs. Extending the command set of the test scenario scripts as well allows him to be able to load all his data to the algorithm when it is necessary.

3.7 Other supportive environment modules

We have covered the whole environment for creating automated poker players in on-line casinos from the front-end and the recognition process, the core of the system, to the way how to implement the back-end and a poker algorithm, and how to test most of the functionality, especially the poker algorithm. At this point the

reader should be able to understand the project ideology, or even more, make his own environment based on the ideas and investigations we have provided without any fundamental difficulties. Probably the most interesting variant is to use our environment focusing on the front-end or back-end creation according to his needs. However, there are still a few things to mention if somebody wants to use our environment and take advantages of all the features it offers.

Peripherals emulation

The functionality related to input and output peripheral emulation and observation is implemented in the *UserIOEmulator* module. It implements methods for screen and partial screen capturing. The results can be used for screen recognition and activity monitoring. In this module we have also implemented an emulation of a keyboard and a mouse. This includes simple to use methods, emulating the mouse clicks, mouse drags or keyboard typing.

Logging

During the whole system runtime it is necessary to inform the user about what is going on, eventually what failed and what succeeded. This is the main concern of the *Messenger* module. It implements methods that can be used in any part of the system to display messages, warnings and errors. However, the module itself is not capable of displaying any message, therefore another object that can display or save the messages must register itself to the messenger. This object must implement the *IMessengerListener* interface. Usually a client provides a text box with these messages for the user and it is the right candidate for the messenger listener.

Front-end implementation

We have mentioned most of the functionality necessary for the front-end creation, but in a general way. The programmer might be a little bit confused when trying to modify or implement other front-ends or front-end parts even though he has the idea and knows how to execute it, the exact implementation and approach were not mentioned in sufficient detail. To make the front-end programming as simple as possible we have created an additional module with encapsulated functionality at one place. It is not necessary to use it, it is possible to work directly with the already described components and system functions. But its purpose is to lighten up and speed up the implementation process and allow the programmer to focus only on specific relevant parts.

The name of this module is *FrontEndHelper*. It encapsulates all the front-end related functionality to one place and adds even more. It is not necessary to take

care of paths to every trigger and recognition manager, nor is there need to take care of trigger action area and subsequent call of peripheral emulator with given value. For example, clicking every time at the same place of a button in a poker client is not wise as well. All this is managed by the *FrontEndHelper*. The actual front-end programing using this helper module looks like helper get this and helper do that. The front-end designer does not need to know anything about peripheral emulation nor trigger manager to create a full-valued poker client front-end.

To create a front-end helper object we need a path to related trigger and recognizer parent folder that is required as a parameter. From this point we can ask for every type of trigger we want to use, just using its name. The helper does the rest. A smart thing it to create multiple helpers, one for each managed window. This way we can separate triggers and recognizers used in different window types. Moreover, this allows the triggers to work correctly and independently on the actual window position. The only thing we need to tell the helper is the position of related window on the screen. Afterwards we just need to call the *UpdateScreenShot* method every time we want to update the related screen data.

When using the helper object we have to keep in mind the synchronization. Not only the output resource synchronization but even the screen capturing synchronization. If there is one global window manager managing for example the window positions, the concrete window managers can not perform any task until the new window positions are set to respective window front-end helpers. The same applies in opposite way, do not touch anything from outside, until the internal window management finishes. Of course, parallel screen processing of windows on different locations is possible.

In contrast, the opposite approach would be to only let a central authority, the window manager, manage the screen capturing. The central authority can assure that each window is using the right screen shot and right window position. This could be ensured by a single method that takes a new screen shot and position. The drawbacks are that the front-end must operate directly with the screen data representation and configure the respective helpers with it. This is still not that bad compared to the fact that a window can not update its screen shot without asking the window manager to do so. This was the way how we have initially implemented the front-ends, but we have changed the helper object to the first approach. The code is cleaner, but the correct update of window positions is not guaranteed globally, instead it is let on the front-end programmer.

Client

The client is a user modifiable gui application. It will provide the basic functionality to run the whole system. Most of the information concerning the client were already mentioned in the front-end and testing chapters. It is possible to choose a front-

end we want to use in the “FrontEnd” menu and configure it through the front-end settings or the front-end mini console. Front-end settings can be turned on in the “Window->Settings” menu. The client will allow us to run the system with test creation support that can be turned on in the “Test->Create test scenarios” menu. In the Test menu it is also possible to run the system in the test mode using the tester application with appropriate poker client interface. The “Action” menu allows us to attach to the poker client, start and stop the play or turn on or off the automatic play.

Summary

At this point we have finished the ideas and approaches concerning the environment for creating automated on-line poker players. What was said is already implemented and waiting to be used in a real situation. In chapters 4 and 5 we are going to employ every part of our environment to create a functional demonstrative Selfacting Internet Casino Entity. (SICE) We will build a front-end for a real casino to prove the capability and strength of our system based on monitoring and graphical recognition of the screen. Afterwards, when it will be possible to plug in the front-end and gather a real table data from real situations on poker tables, we may start the back-end implementation. Here we are really inquisitive how our approach to algorithm implementation holds up. Of course we are going to implement the basics strategies for the no-limit tournaments. After all, when the whole system will be up and running, we will try to answer our target questions and give a summary.

Chapter 4

Building a casino front-end

In this chapter we will try to find a suitable casino and build a front-end for it using our environment for front-end creation.

4.1 Terms of use

We are going to search our representative set of casinos for a suitable casino for this thesis. We must take into account their usage policy and find a way how not to break the agreement rules while implementing our automated player. Nevertheless, this does not mean that it will not be possible to build a poker robot working on these sites using our environment. If we succeed in this thesis with any casino that allows to operate an automated player, somebody else can do the same in other casinos and disobey the conditions of use. However the anti cheating methods used by poker rooms differ and other additional techniques to stay hidden when operating a robot might be required. Let us begin with the biggest and best known poker sites like PokerStars, FullTilt, TitanPoker and PartyPoker and see whether any of these can be used for our purpose from the usage agreement point of view.

PokerStars

PokerStars is known for the best detection of robots and any suspicious software of all poker sites. Most of the competitors' unfair software claims difficulties when operating on PokerStars and they require techniques to operate their software from a different machine using desktop sharing for example. The terms and conditions of use, however, are surprisingly very mild and suitable for our needs. It is allowed to use other software to operate their poker client which is very important for us. The user, however, must personally initiate the action without any additional artificial

advices. This is claimed in the third party tools and services[1], and in the license agreement[2].

FullTilt

FullTilt poker does restrict all kinds of player assistance programs that gives the user an unfair advantage[3] over others. This includes only software and databases containing more information than the user obtained or observed by himself. The robot play is handled explicitly[4]. All interaction with the poker client must be executed personally and that their user interface must be used. This is pretty much the same condition as in the PokerStars. FullTilt is therefore suitable for this thesis in the same way as the PokerStars is.

TitanPoker

TitanPoker is "the king" in strictness of using their software. It does not allow to "interfere in any way with the Poker Room (downloadable software) or the Website (their web site)".[5] Although the context is the robots and similar devices, the clause itself in its pure meaning restrict even human operators. One way or another any automated interaction with the poker client is not allowed on TitanPoker and we have to pass by.

PartyPoker

PartyPoker does restrict the use of software programs that are designed to enable any artificial intelligence. They do not have any particular sentence in the terms and conditions of use regarding the automated operator of their client. From this point of view this poker room is suitable for our purpose unless any "artificial intelligence"[6] is implemented and used to our advantage. Likewise it is questionable whether a series of rules and a random number generator can be considered as artificial intelligence. Is is not prohibited to automatically compute some auxiliary values. Is this artificial intelligence? As the author of the algorithm rules we are able to act according to these rules even without any software tool. The problem

[1] Available on-line at *http://www.pokerstars.com/poker/room/prohibited/.*

[2] Provided with the PokerStars software, see the clause 5.5.

[3] For the exact definition of an unfair advantage see the clause 8 in their site terms provided with the FullTilt software.

[4] See the clause 9 in their site terms.

[5] See the clause 4.1 in their site terms provided with the TitanPoker software.

[6] There are multiple definitions of this term. For more references see the Wikipedia. Artifcial intelligence - wikipedia, the free encyclopedia, 2009. [Online; accessed 16-April-2009].

we might walk into when using this poker room is completely different. They will take measures to detect automated players by screen scraping of the user computer[7]. This is something we do not want to undergo due to our privacy, nor due to possible problems during the SICE development, even when not breaking any agreement rule. By the way, we do not see a point in this approach, the robot application can be hidden in the background without any graphical representation on the screen.

We have found two major poker rooms that fit our needs sufficiently. We were not expecting to find any commercial poker site that allows automated players. However, being able to observe the course of events and being able to control the poker actions from an external application is favorable enough. Let's pick up one casino for this task, PokerStars is offering itself as the biggest and most widespread poker room of all.

4.2 Closer look at PokerStars

The "winner" of our quick juristic survey of the usage conditions in a few poker rooms is the PokerStars with FullTilt right behind. We have preferred PokerStars to FullTilt due to their bullet-proof anti bot policy, at least presented to the public and claimed by concurrent robot developers. We are interested in whether even a complete emulation of a human behavior, as we are going to implement, can be detected. Let us take a look on what we can afford to do in PokerStars, as we know that building a full operational robot is not allowed anywhere. The information is taken from the third party tools and services available on their web page. The following tools and services are explicitly allowed:

- Tools, services or charts that simply tell us odds, starting hand recommendations, etc.
- Tools and services that profile our opponents we are playing with.
- Macros and Hotkey programs that don't have any bearing on gameplay logic.

Especially the first and the third point are very favorable for us. We can gather, compute and display any information regarding the table state. It is even allowed to compute and display some back-end specific data used for our algorithm decisions. The third point says that we can use our program to perform an action on a table. Emphasizing that it is us who can perform the action, not the algorithm or the system itself. The prohibited services and actions are:

- Colluding – sharing card data with other players.

[7] See the clause 7 in their site terms provided with the PartyPoker software.

- Working with central database of player profiles or hands.
- Program that plays without human interaction.
- Practice data-mining – observing games as a non-player to build databases of hand histories.
- Any software that offers direct game play advice on the appropriate action to take, and which is also programmable or configurable beyond a very basic level.

Only the third and especially the last point affects us. The third point says what we already know, that we need to make the action personally. The fifth point is questionable in our conditions. We want to build that tool as a demonstration that it is possible to do so. We do not want to use it or rely on actions it wants to perform. The same logic as we have used with the artificial intelligence in PartyPoker applies here similarly. Is it an advice to see "go all-in with those kings" if we programmed it and if we know exactly what the "advice" will be?

As we are not making a thesis on philosophy or law and we do not want to argue about this let us assume the following scenario of use. The table is observed by our front-end and all the necessary information is extracted. Subsequently the information is sent to the back-end to compute the resulting action. The client catches this action and gives us a choice of performing a move by emulating the same poker client interface. We can call this interface "hotkeys" for folding, calling and raising. After we have decided for an action and performed it, the client shows us the action computed in the back-end for comparison. As long as we are able to predict the requested action everything should work fine. If these actions do not match, then we or the back-end made a mistake, we can correct it, forget current hand and continue with a next hand.

The final step would be to interconnect the action from the back-end with the front-end directly. This is something we are not going to do due to the agreement rules, but doing so is a trivial thing and the system can be considered as a self-acting entity capable of playing hold 'em poker on PokerStars which can be considered as fulfilling one of the goals of this thesis.

4.3 PokerStars front-end architecture

We will use the same architecture as described in section 3.2. A *WindowManager* will be used to handle the overall window placement and desktop organization. We will create two concrete managers for specific windows, one for the main lobby window and other for a poker table window. The *LobbyManager* will be responsible for handling the operations with main lobby such as registering to appropriate

tournaments. The *TableManager* will be capable of creating the table data representation based on actual table image. In our PokerStars front-end we will have one *WindowManager* and one *LobbyManager*. The number of *TableManagers* will depend on how many tables we are playing simultaneously. Each manager type will require its specific recognizers, triggers and configuration data. We will create a "data" directory in the project with well-arranged structure holding all related and necessary data.

4.4 Creating recognizers and triggers for PokerStars

We will create a separate recognition set for each manager used in our PokerStars front-end.

4.4.1 Poker table recognition

At first, we have to make a few observations of a typical PokerStars table which is depicted in figure 4.1. Although we know that we want to build a front-end that recognizes the lowest no-limit tournament turbo tables, later or even during the testing phase we might want to use the front-end on other tables as well. Therefore it would be the best to create the most universal selection of triggers and recognizers that can be reused on different tables.

PokerStars tables look pretty much the same, it does not matter whether we look at a cash table or a tournament table. The only difference is in an unimportant piece of graphics in the middle of the table. The important difference for us is the number of available seats and their positions. There are tables for ten, nine, six or two players, probably there are some more which we have never noticed. The six and two player tables are derived from the nine player table with some seats excluded. The tournaments with the smallest fees take place on the ten player table version, tournaments with higher stakes are held at the nine player version of a table.

The picture above is the ten seat version of a table with players 3 and 5 already out of the game. Let us create all the necessary recognition data for this table type, the same process can be used to create the recognition for the nine player table version. To make our references to the picture simpler we have named the players in our own way. This will also be our referential order of players with the first player in the upper right and the last one in the upper left.

Figure 4.1: PokerStars table

The presence of a player at a given seat, dealer button position, or whether a player is still holding his cards can be determined using a pixel trigger with specific pixel recognizer. For example, a corner of the box around each player name, a red pixel in the 'D' of a dealer button or a specific pixel on the mini cards laying on the table upside down can do the right job. In each of these cases we create ten pixel triggers sharing one pixel recognizer with specific pixel color. This is everything that is needed. After all, the most time-consuming thing will be to wait a whole round to get all the dealer button positions into the trigger editor. At the end, when we have captured all positions and placed them in order in the trigger editor we can create a pixel trigger list for the front-end.

To figure out whether it is time to act we can watch the fold button. Every time it appears we have to recognize the whole table and create the table data structure. Sometimes a "muck hand" button may appear at the same position, therefore the pixel recognition is not sufficient in this case. An image trigger and image recognizer of the fold button is the right choice. We also have to declare the areas of other buttons and the raise text box to be able to press them, or to select the raise amount and replace it with our value.

The rest that remains to be done is the font recognition of stacks, bets, pot and cards. The same font is used with player stacks and the pot, therefore we could spare one font recognizer for pot recognition. Any tournament in a later phase or a cash game will be suitable for creating the font recognizer as many different stacks

with different numbers are present. We will also need to pick up players that are sitting out, disconnected or all-in to be able to correctly recognize their state. Of course, we have to include the punctuation, dollar and the word 'POT' if we want to recognize the pot with this recognizer as well. The rest of the job is on the text triggers to correctly adjust the font recognition parameters as position and search distances, plus override the colors in case of the pot recognition.

We should make a remark on user customizable triggers, which were not mentioned yet. It is possible to create a customizable trigger for special occasions when the trigger values are not known during the trigger creation. By inheriting a class from the original trigger we may modify its internal values and customize it during runtime.

The pot recognition was the case when the customizable triggers came handy. The pot font uses the same color as the box around the yellow pot background. The size of the background rectangle is adjusted to the size of the pot text length. This could confuse the pot recognition as the box may be considered as a character that is not known to the font recognizer. We had two options. To add the horizontal line to the font with empty output character or to make an adjustable text trigger and set the actual parameters respecting the size and position of the pot text during runtime. To find the pot brims we have used two image finders, one for the left and the other for the right edge of the box.

For the bet recognition we have made a separate font recognizer. The only thing we had to take care of was the correct creation of appropriate triggers. The position of the bets may vary according to number of chip piles. This position is sometimes shifted to the left and sometimes to the right, depending on the seat position and the direction in which the chip piles grow.

To recognize the cards we have created two fonts, one for the card number recognition and the other for card suit recognition. In fact we do not recognize the card color but the shape of clubs, diamonds, hearts and spades, each as a single font character. Although it is possible to do it through the pixel color recognition, this way is more elegant and does not require a separate trigger for each color. Afterwards we have created text triggers for every possible card location around the table and on the table. This might seem a little bit unfit compared to a trigger grouping approach, which means instead of creating a recognition for every single card, create a recognition for a hand and use only single position for hand recognition. On one side we may spare some triggers differing only in the position, on the other hand we need to provide relative shifts between a card and its marks. The overhead from creating a custom hand recognizer is not big, but the payback is not significant either. Maybe if more visual elements such as the dealer button, presence check location etc. could be grouped together to create a complete position recognition trigger. In any case, the trigger editor is simply doing a great job in the mass production of

appropriate triggers.

There are three or four ways how to recognize the blinds and antes. The easiest is to parse them from the window title bar. This makes the front-end dependent on the font we use in the system, however what is more fundamental is the fact, that the blinds may not fit into the title bar when the table windows are diminished. Another approach can be the notification observation that the blinds are increasing, or a time measuring knowing the blinds structure and time interval of increase. The drawbacks are obvious, we may miss the notification if the table capturing frequency is low or suspended for a while. Synchronizing a time with a tournaments will not work as the tournaments can be suspended for a while due to some other reasons. The last approach is to compute the blinds and antes from the stage. We have decided for this approach because it is reusable in other front-ends although a lot of specific situations had to be handled. These situations are, for example, that we do not see the scene right after the blinds are posted, but rather when it is our turn with a lot of action already done. The blinds might be so high that a player might go all-in and not even reach the blind size. Antes are paid even by players that already folded hands etc.

The last requirement for a table recognition is the need to be able to detect a finished tournament. In PokerStars this is done through a message box dialogue saying that the tournament has finished. We have to detect them and press the appropriate button. This is an easy job for an image trigger. A thing to remember is that other messages can be displayed as well, for example the message about transfer to another table when playing multi-table tournaments.

4.4.2 Poker lobby recognition

There is far less work with the recognition of the lobby when compared to table recognition. The only job of the lobby manager is to automatically register for a tournament when requested. It needs to be able to locate the right tournament in the tab structure, find it in the list of tournaments, pay the tournament fee and register to it. Once again, it is no deal for image triggers and button pressing in the right order.

4.4.3 Window manager

The window manager has to ensure that every observed window has its unique position on the screen. It is also responsible for picking up all poker-room related windows and incorporate them to the overall scenery. This applies to the system startup in case there are already some tables present and also applies during the system operation when new tables are opened and finished tables are closed. The

window manager is also responsible to create the lobby manager and the table managers on appropriate windows.

The localization of the related windows on the screen is done through the Windows taskbar icons. Using an image finder it is easy to locate the corresponding poker room buttons. Once the Windows taskbar was located it is not necessary to search through the whole screen again. The search area can be reduced to a small square or even a line at the right position. The window control is done by the taskbar menu commands available through the right click on the respective button. We are able to move, close or restore a window. This approach is dependent on current Windows scheme which is not a very proper solution, but can be easily adjusted on both sides. Using the right Windows scheme or using the right recognizers and menu position of requested actions.

The window manager has to regularly wake up all the table managers and let them check whether it is necessary to make a move and perform the table analysis. The same is done with the lobby manager if a new tournament should be started.

4.4.4 Implementation techniques

The whole manager implementation is hidden behind the *IFrontEnd* interface. Every front-end request is delegated to the window manager for synchronized processing. We will make a few remarks on the front-end structure. Every manager uses its specific data. These data are loaded to auxiliary objects called *DataHolders* and *TriggerHolders*. These objects can be created using the appropriate path to configuration data or with an appropriate front-end helper object. They load and create all the necessary data that can be later used directly from the managers without any additional on-place loading and error handling. This also ensures that a variable with the same name can be used in all managers to access only relevant data and triggers of each manager, which is very transparent.

The PokerStars front-end operates only with one game type at once. We have not implemented a game type recognition of a poker table. The front-end assumes that every table it faces is of a given type. Of course, this type can be changed through the front-end mini console. This only affects the automated joining to the tournaments and initial process of attaching to the lobby. Only the tournaments of actual type are joined automatically and played. Indeed the game-play on already opened and monitored tables is not affected by the game type change. It is possible to start a small tournament for ten people, then change the game type and start another type of tournament for nine people only. Both will be monitored and correctly recognized.

4.5 Client implementation

There is nothing special concerning the client implementation. It is a simple GUI program and everything what it does was already mentioned. The source code of the client is intended for the user modification too. He may add more front-ends to the supported list of front-ends or implement any specific interface for controlling other system parts in case that the front-end mini console approach is not sufficient or suitable enough.

During the client implementation we have also created a test back-end, named *testAlgorithm*, to be able to run the system for the first time and see whether the front-end gathers all required data correctly. The test algorithm creates only a random action request which should be changed by the user in the client. This action is subsequently executed by the front-end on the appropriate table. For the first time we are able to play poker on PokerStars using an external application providing all the necessary data and a user interface for resulting action. With the help of *TestScenarioCreator* we were able to start building our set of test scenarios that will become handy during the algorithm implementation and testing.

4.6 Poker Academy Pro

In advance we will need to admit that implementation of the back-end and the no-limit tournament algorithm using the "guess what the algorithm do" approach was very uncomfortable. In some corner cases it was difficult to predict the exact action of the algorithm, especially when a complex and complicated computations started to appear in the algorithm. Without knowing the answer on whether we are able to see the requested action before we approve it this was not a good idea how to continue. Therefore we have decided to leave the PokerStars front-end for a while and try another solution of this situation.

Its name is Poker Academy Pro, a software designed for poker training, using artificial poker players. We have already mentioned it during the data mining process of concurrent applications. Using the Poker Academy Pro as our referential casino brings these consequences:

- We can use their graphical representation of the casino, not only their *Meerkat API*, to simulate the whole process from recognition to action making.
- Poker Academy Pro can simulate sufficiently enough a no-limit tournament with a little bit configuration.
- Speed up the game play. We do not need to directly intervene to the play and we do not need to wait for other slow human players either.

- We can compare our algorithm skill to their already implemented robots capable of no-limit tournament play.

- Poker Academy Pro is not free and costs $85. The demo version works only for 6 hours of net playing time.

Everything looks good except for the last point. We will have to be really fast and probably use the demo version on multiple computers to create a solid test-case base and do some algorithm skill measurements. This way we will be able to present the whole system playing automatically.

Despite the fifteen years of development done at University of Alberta their main focus is still aimed only at a theoretically perfect fixed-limit cash games. Although they have implemented no-limit tournament players using many different and very advanced approaches and methods, the quality of such players is a little bit controversial. A lot of work can be seen on their tactics, bluffing and trap preparing. Some of the robots understand the need of blind stealing and can adjust their game-play more or less during the whole game. We doubt that we will be able to cover that much area of strategies in our player. On the other hand, these robots do not reflect the situation in real casinos precisely. They tend to make more calls and raises when compared to the real tables. Although these calls and raises are often in conformity with the math, the people in real casinos are more cautious. For a human, or at least for somebody who is trying to implement a poker intelligence, it is very easy to find some threshold values of necessary pressure and aggression to deal with these bots. The pattern of the game play is pretty much the same all the time but in spite of that, it is very instructive for a newcomer. One could say that these bots play no-limit tournaments as a cash game with increasing aggression according to rising blinds. This is of course only an initial impression after a few games we played, to eliminate the luck factor as much as possible and see where their bots really stand we should play more games.

By the way Poker Academy Pro it is worth attention and for a starting player it could be a really good choice where to learn the very basics of the game. Although the money motivated game is always the best teacher.

One conclusion can be made one way or another. These robots are capable of a solid no-limit tournament play and theoretically with a combination with the recognition environment could be dangerous to the on-line poker industry, or rather to inexperienced on-line players. Maybe knowing that the opponent is a bot makes a huge difference between being able to defeat it or to be defeated. It seems that it is definitely possible to make even the no-limit tournament automated players.

The question remains whether we need a huge team and many years of research or whether it is possible to create a no-limit tournament bot with much less effort.

- In this thesis we will focus on making a competitive no-limit tournament bot

using our environment that will be able to keep up with the robots from Poker Academy Pro to figure out how difficult it is and if it is possible.

- As a success we will consider to break the even with the rakes excluded. This means not to be in a loss when we do not count the rakes for a casino. We just want to confront the bot skills on both sides.

Front-end

We are not going to describe the creation of the Poker Academy front-end step by step. The same approach as with the PokerStars has been used. Moreover, the PokerStars front-end source code was copied and adjusted in a few details. The largest amount of work has been done in the trigger editor. It was necessary to recreate the positions, images and texts for the respective recognizers and triggers. A few triggers and recognizers were removed completely, a few were added. For example, the pot handling was changed as in the Poker Academy they use multiple pots as players goes all-in. Although the Poker Academy Pro graphical interface was not included in our representative set of casinos, the recognition system proved that it is strong enough to cover a casino we have never seen before and in a few hours we were able to recognize everything that we needed.

Unfortunately, their graphical interface does not print the bet sizes at all, only the poker chips are displayed and some action is printed next to each player. This is very confusing and very uncomfortable even for a human player. Moreover, they use the "raise of" notation instead of the standard "raise to" notation. These two facts together create an atypical situation. It therefore takes a while to adapt to it and understand what is going on. For our recognition system this meant to deduce the missing and standard information from the pieces provided. This took even more time than the overall graphical recognition, but we have succeeded in the end. Our new front-end is up and running.

We have also adjusted the client to allow auto play and to allow a better user interaction with the requested actions.

4.7 Casinos' protection

Based on our investigation and experiences with the design and implementation of our environment we can summarize some methods and hints that could be used by casinos to make the bot implementation more difficult. Our recognition subsystem was designed to work with current mechanisms used by on-line casinos. It is therefore questionable whether there will be any advantage for the casino to go against this trend and whether there are methods that completely eliminate robots

based on optical recognition. We think that this is a question of balance between what the recognition system can recognize and how many people are still willing to play in given casino.

Our recognition system is based on the fact that every casino uses a static composition of the scene. If a casino can break this presumption we would have significant difficulties to operate in this environment with our software. This can be done easily by modifying the positions of visual elements. However, this could be very unpleasant for human players. Another way of breaking this rule would be to use transparency, putting multiple objects one over another in overlapping positions and preserving the component positions. This scene would still be readable for a human, the bot, on the other hand, might experience significant difficulties. To minimize interventions in the relevant components it is possible to preserve unique locations of each component and use the background or other unimportant objects to interfere with them.

Another idea on how to break the recognition process is to use smoothing and antialiasing with these items. This could only be applied when using big versions of tables and might become unreadable on smaller tables[8].

The casinos have, however, adopted more powerful and we believe better methods to protect against the bots. One crucial presumption they can apply is that only profitable players can be operating a bot. This significantly decreases the number of potential invaders. When somebody is playing for a long time, the casino contacts him with a special dialogue, requesting a fundamental action that a bot cannot make to be performed. However, some threshold values of continuous play that will not be suspicious can be found for each casino.

[8]For more information see appendix A and attached casino table examples.

Chapter 5

Building the back-end

The last thing that remains to be done and the last question that stays open. The back-end and more importantly the algorithm inside. The question whether it is possible to create a decent automated no-limit tournament player with affordable resources. Anybody who is really serious in creating a poker robot that will be earning money for him will not have any problem with using other forbidden methods. He will probably choose the most suitable version of poker, the fixed-limit cash games, as these are much easier to compute or train on a huge hand database with adequate actions. I would not be surprised if the guys from the Poker Academy Pro software were using their bots at home playing on-line, but even if, they would probably use their much better fixed-limit cash game bots, not the no-limit ones in the tournaments. Let us take a look whether we are capable of creating an intelligence strong enough to compete with humans or the best robots available in the no-limit tournaments and answer the question whether we may face a robot in a real casino even on no-limit and tournaments.

We will implement the back-end and its algorithm in the exact way as it was mentioned, using the *AlgorithmHelper* module and state automaton approach together with the *HoldemPoker* module and its supportive functions.

We will also use a combination of both, PokerStars and Poker Academy Pro software for our algorithm development. In the early phases with no significant intelligence implemented the PokerStars and some small real money or play money tables fits better. In the middle and later phases when the algorithm will be capable of making whole range of decision and will be able to play independently we will use the demo version of the Poker Academy Pro.

5.1 Target tournament type

We will try to develop our algorithm for general-purpose, but the specific settings and adjustments will be pointed at a single table sit and go no-limit tournament for nine to ten players in the turbo variant with the lowest or very low stakes. The lower the fees the worse players are around in general. The turbo tournaments were chosen because of reducing possible observations of betting patterns that can be made by human players, but not by a robot, at least not in this version of our algorithm. However, this could be a way of improvement in the later phases. For a real robot usage even the higher tables should be considered. There are lower entry fees to the casino compared to the total prize pool, on the other hand, we may expect better decision making of other players. Unfortunately, we will not be able to test it on a sufficient number of games with a fair usage of the casino to make any statistics on this.

This could be a very interesting, however illegitimate, investigation. Having the same algorithm and let it play on different tournaments, on different time of a day or night, and on different days. Or even in different casinos. As a result it could be possible to point out the casinos having the worst or the best on-line players or to choose a right and wrong time for playing. Although this might be related to the algorithm strategy and worth only for players with similar game-play style.

In our case we will use the Poker Academy Pro as we pronounced. We will need to adjust some settings to emulate a tournament that most resembles the PokerStars tournament. We will use "UB – Low Buy-in" template for the basics. Afterwards it is necessary to copy the blind structure and set ten players and a single table mode. The most important step is to change the blind rising method from "exponential", when the blinds are rising after a specific number of hands, to time dependent. We can use one minute for blind rising and adjust the bot reaction speed accordingly in the tournament to represent a turbo tournament in a standard casino. In Poker Academy Pro there is a preset opponent structure for a no-limit tournament called "No Limit Standard", we will use this for the beginning. We have kept the tournament fees at $5 entry fee plus $0.5 rake for the casino. Now we can enjoy an emulated no-limit turbo tournament for free for a while without any limitations.

5.1.1 Standard turbo tournament settings

In the algorithm implementation we will assume some tournament settings. We do not hard-code these settings into our algorithm, we are mentioning them for better reader imagination when we are going to talk about specific values and tournament phases. For a standard tournament we will choose the settings used in the PokerStars tournaments, there are no huge differences in other casinos. The ratio of all related values is very similar and creates a tournament with alike structure.

The usual payout structure of these tournament for nine to ten people is 50%, 30%, 20% of the overall prize pool for the first three players. The additional fees for the casino are paid extra and vary somewhere between 10 to 15 percent. On PokerStars the lowest turbo tournaments are $3 + $0.40 for ten people with the prize pool of $30, or the $6 + $0.50 with nine people and prize pool of $54.

Every player gets $1500 in chips at the beginning. These chips have no real value, the only important thing is the place when a player runs out of these chips or owns all the chips at a table. The blind structure is depicted in the Table 5.1, with increase to next level every 5 minutes.

Small blind	Big blind	Ante
10	20	0
15	30	0
25	50	0
50	100	0
75	150	0
100	200	0
100	200	25
200	400	25
300	600	50
400	800	50
600	1200	75
800	1600	75

Table 5.1: Blinds structure

5.2 No-limit tournament algorithm

The approach to the tournament algorithm is significantly different when compared to a ring game. This applies especially in the turbo variant. We simply do not have enough time to wait for a great hand, we need to take advantage of other factors and situations. There is not enough time to read our opponents well enough but we don't have to worry about making our play style transparent. This is in favor of an algorithm.

The no-limit tournament algorithm we are going to implement will cover the fundamental actions and strategies. Most of these strategies are covered by Harrington on Hold'em - Volume I[1]. We are going to add the functionality and algorithm skills according to the benefits that they can provide in the long run. After a series

of improvements and added skills we can test the results on a real tournament and eventually make additional improvements and fixes. During this development we will create a solid base of test scenarios to make the development and testing as efficient as possible. At the end of this development process we will hopefully attain a reasonable playing intelligence and prove that it is possible to build an automated on-line no-limit tournament player using our environment and state automaton approach.

Before we start with our algorithm implementation, we shall just make a few notes about observing a table. At real tables it is essential to observe other players' ticks, hand moves, or the way they play with their chips. In on-line tournaments, this is simplified to only observing betting patterns. We will discuss possible observations that can be done to improve our algorithm game-play later. Another rule that applies to on-line tournaments is to bluff much less compared to real events. A lot of people playing on-line many tables at once and are watching TV besides at the same time, which makes it hard for us to create an image of a conservative player. This is especially true at the beginning of a tournament. However, as the number of players at the table shrinks, these observations start to be made more easily.

5.2.1 Overall algorithm strategy

Our algorithm will be capable of playing the whole range of strategies from extremely tight to very aggressive. This will depend on the phase of a tournament and will be changing during the whole game. In general, the aggressive style requires more skill and a better play. This is mainly caused by the fact that aggressive players get to situations where they face harder decisions and have to read their opponents well. We will follow the next rule:

- Make that kind of decisions which make the subsequent decision making as easy as possible.

This rule should be also adhered by humans. A player should avoid situations in which he has no clue about what cards his opponent could have. Therefore, a player should adjust his early decisions to simplify his later decisions as much as possible. Our algorithm will sometimes sacrifice potential profit in the current betting round to avoid harder decision making in the future. We take this approach because harder decision making in the later rounds of betting usually implies a potentially higher loss of chips. This means we are sometimes going to bet more or less than the "correct" amount in order to clarify the positions of both us and the opponents.

As an example we describe the following situation. Someone opens the pot with a raise. We have pocket queens. A re-raise would be appropriate. What actions could we face later? Another re-raise, an all-in move or a fold are the good ones.

Here we just have to decide whether we believe that the opponent has pocket kings or aces and see how much it costs to call compared to the pot. If the opponent folds we collect the pot without any further action. It is however more likely that we will face a call. This puts us in a difficult position on the flop. The chances of flop containing an ace or a king that could beat us are somewhere between 37 to 48 percent, depending on how many of them the opponent holds in his hand. Moreover, we should be afraid of a jack or a ten, which might help our opponent make a set if he was raising with the appropriate pocket pair. In most of the situations we will face a though decision what to do. The question is, isn't it better, easier and even more profitable in the long run for an algorithm to go all-in on pre-flop instead of raising? We will stay away from most of the problems and tough decisions. The theoretical loss caused by slightly imperfect play may be acceptable and still cheaper than committing our entire stack to somebody who hit his card even with weaker pre-flop hand later.

5.2.2 The M-ratio concept

Simply, the M is a number that defines how long we are going to stay in a tournament without any action. That means how much time we could afford to wait for a hand with which we decide to play and try to win the pot. It is obvious that with shrinking M the need to make a move increases because of the possibility of being blinded-away from the tournament. However, some types of play might become less attractive because our maneuvering options are becoming limited. We need to adapt our game style according to the M-ratio. In conclusion, the lower the M is, the more aggressive - and of course risky - our play should be.

Another angle of view on the M might be how likely we will get a better hand in situation still with a reasonable amount of chips to play. Many times it is better to undergo a higher risk with more chips than play a monster hand with just a few chips left. While in the first case we may be eliminated more likely, we will get a solid stack for further play if we win the hand. On the other hand, if we double-up with our super strong hand but only with a few chips in front of us, we will face putting ourselves at risk once again in a while, but often before another monster hand shows up.

Harington in his vol II intimated Magriel's M-ratio concept on some examples together with his so called Q concept as the weak force[2]. Further he extended the concept for short tables and takes into account the shrinking number of player mostly for a single table tournament. It is called the M effective[1]. For our purpose we will adopt this concept and use the M effective referenced as the "M" from this moment. The definition of M is:

[1]This is being discussed in Harrington's Volume II from page 277.

$$M = \frac{player\ stack}{small\ blind + big\ blind + total\ antes} \times \frac{actual\ players\ num}{total\ players\ num}$$

Based on M, which projects the overall tournament situation and progress to a single value we are able to build a robust strategy for every tournament phase.

5.2.3 Harington's zone system

According to the M concept Harrington has developed and written an overall no-limit tournament strategy that respects the need of play-style adjustment during the whole tournament. The lower the M is, the more aggressive style is required. Harrington distinguishes a few zones, each representing an interval of his original definition of the M-ratio. The tightest strategy is described for the green zone, with M more than 20. It then continues through the yellow zone with M between 10 and 20 followed by the orange with M still above 5. The last is the red zone with the most aggressive strategy for M less than 5, or eventually the dead zone for M less than 1 when there is no strategy required, the only hope is to be lucky. He described his strategies for each zone in consideration of a multi-table tournament. This could be a very good base for our algorithm's implementation. The overall idea and the skeleton is great, the particular zones and actions must, however, be changed and adjusted for the algorithmic approach considering the adjusted M definition for single table tournaments.

5.2.4 Hand sets

To be able to cover all the possible hands that might be dealt we need to separate them to hand sets with similar pre-flop strength. In the algorithm we do not want to evaluate every hand or a card that might be suitable for some kind of action separately. We will decide for a pre-flop action mostly according to the hand set where our actual hand belongs to. A good base for hand raking and hand division into groups was described by David Sklansky[3]. He used eight hand groups numbered from one to eight with similar pre-flop strength, or rather more important with similar pre-flop strategy that should be used with each of these groups under given circumstances. We are going to adopt Sklansky's hand sets and adjust them to our needs a little bit. Our sets are declared in the Table 5.2 ordered from the best hands to the worst. The remaining combinations are not included in our hand sets, they will be referenced as the rest.

Hand set	Hands contained
ABest	AA, KK, QQ, JJ
APlus	AK, AQ, AJs
A	AJ, KQs, TT, 99, QJs, KJs, ATs
BPlus	JTs, KQ, 88, QTs, KTs
B	AT, T9s, 98s, J9s, 77, 66, 55, KJ, QJ, JT, Axs
C	T8s, 97s, 87s 76s 65s, 86s, 54s, J8s, 75s
D	KT, Q9s, QT, 44, J9, T9, 33, 98, 22, Kxs, Q8s, Ax, K9
E	64s, 53s, 43s, T7s, 87, Q9, 76, 42s, 32s, 96s, 85s, J7s, 65, 54, 74s, T8

Table 5.2: Hand sets

It might be useful to also define multiple pre-flop hand sets in one algorithm. This could come handy in some specific situations, for example the heads-up. A hand set with small or middle suited connectors looses its value with shrinking number of opponents. Hands in this group are very weak but might become monsters if they hit a flush or a straight. Such hands are good when it is possible to see a flop with many opponents cheaply, hoping for a monster hit. Against a single opponent they are usually beaten by a higher card.

In our case this is the C hand set. It mostly contains the drawing hands better against multiple opponents. But the D hand set is definitely better against one or at most two opponents despite being ranked as worse. Instead of creating multiple hand sets we will explicitly exclude the C group in the algorithm in some cases, when the D hand fits, but C does not. Compared to the Sklansky's groups we have modified especially his Group 6, which is our C hand set and as we have created the drawing group. This was done due to the algorithmic approach. It simplifies the number of decisions that the algorithm must cover. When seeing a flop with a C hand it knows it should focus on the straight or flush draws. If it hits a top pair it is usually not good enough because of a weak kicker.

5.2.5 Algorithm zone system

Our algorithm will generally adapt five ranges varying on exact situation. These ranges will represent the mentioned Harington's zone system. With combination of the hand sets we have created it will be relatively easy to describe our strategy based on the M-ratio for every possible card combination we can get. For a given M range we will define pre-flop actions for each hand set. With the amount of hand sets we have created we still get a reasonable number of basic states and strategies we need to handle in our algorithm. In addition, by this approach we could be pretty

sure that we have taken a stand in our algorithm to every situation that can arise in a no-limit hold 'em poker.

Sometimes, the M-ratio alone is not sufficient enough to make a competent decision. There is another element that can have significant impacts on our decision making, namely *the position*. Obviously, there are many positions where a different approach is necessary. The positions we are talking about are:

- big blind position
- small blind position
- dealer position
- early position
- middle position
- late position

These positions are defined considering a full tournament table with nine to ten players. However, only on a few of these positions a significantly changed strategy is inevitable. A special handling of play is definitely required by the small and the big blind positions. These are the positions that are very likely to face attacks from other players, therefore some increased resistance is necessary. This could be based on the M-ratio concept as well, we need to fight for the chips more and more with the M being lower and lower when somebody tries to steal blinds from us. Handling the button and the late positions is the opposite sides of a coin. Here we need to try to steal the blinds with more hands that we normally use for a raise in earlier positions.

We will merge all the other positions except of the blind positions under one category and try to handle them in unified approach. The changes in game style depending on whether we are on an early or a late position is not that much significant, although still crucial for a good play. We can easily handle these differences in position by using additional queries on our position, number of players that already joined the pot and a potential number of players that could move into after our move. According to these additional queries we can adjust our strategy if necessary, but we avoid implementing very common strategies for all possible positions around a table. Moreover, the algorithm designer can decide to create multiple states with different strategies for early, middle or late positions if he needs. The idea of covering up all possible situations that may arise preserves one way or another.

The final note regarding the global algorithm approach that needs to be made refers to the final stages of a tournament. As the money is within grasp, another set

of factors may influence our tactics. In a situation when there is the last player without any money to be knocked out, another arsenal of weapons is suddenly available and waiting to be launched. If there is, for example, a short stacked player and we are in the position of a big stack, then we can easily steal chips from the rest of the players. Nobody wants to compete and put himself at risk of being eliminated without a single penny when there is already a predestined victim to be eliminated from the tournament. This applies even to very strong hands and works vice versa as well. Sometimes it might be right to even fold pocket aces and wait until the short stack is wiped out. It might be useful to handle the final stages of a tournament in a completely different set of states and approaches, especially the bubble and the heads-up, but still under the M-ratio concept.

5.2.6 Algorithm implementation

We will divide the algorithm to multiple source files, each handling one of the specific situations described before. These are the blind positions, heads-up and other cases. In the basic pre-flop state we distinguish between these situations and the adequate state, or rather a state procedure, is called. Actions are computed according to the M-ratio concept in all of these situations. We will discuss the M based zone system generally and will mention the differences for every situation in respective zones. Our explanation style is the opposite of real algorithm implementation.

5.2.6.1 Folding, checking, raising

These are the things we can not avoid in algorithm implementation. Just a few comments on how these actions should be interpreted when are played by the algorithm.

Fold: The potential profit with actual hand is not worth the size of a call we need to make.

Check-fold: Our hand is not strong enough for putting in additional chips, but if nobody makes a bet we can go on for free.

Call: When we believe that our hand is good enough and worth staying in, or when we have a very strong hand and plan to raise or re-raise later without making any attention now.

Raise: This action was the most important reason for creating this subsection and deserves to be described in more detail. A raise could mean that we want to win the pot as it is without any fight. The other option is we might want to get rid of

the drawing hands or be paid by the drawing hands, or we may try to put in more money to build a big pot with our strong hand.

In real poker, usually the second case with drawing hands is important. It is about balancing on the edge between giving others bad pot odds, but not to scare them away from calling us. The difference between the right odds to call and the odds we have offered and get called is the profit, or loss of course, if we play badly. In small tournaments, which we are interested in, this does not work very well. When we want to make a move, we usually want one or at most two callers with some solid hands that we are going to beat. On these small tables, however, making the right bet does not discourage others from calling us which could result in a situation where our pocket kings stand against five callers. This is exactly what we do not want to happen.

To achieve this effect in a low buy-in tournament we usually need to over-bet the right raise amount. However, it is really difficult to figure out the correct amount. Sometimes it needs to be just a little bit more than usual, sometimes half of or stack. For the algorithm it is almost impossible to guess the right amount because it depends on the constitution of the table and the players around. A lot of observations and player reads would be required.

Luckily, there seems to be another way how to handle this situation inspired by the Sklansky's all-in or fold pre-flop theory[4]. As the name prompts, making an all-in move instead of a raise is the key. Not only we honor our rule of making further decisions as easy as possible for one hundred percent, but the vision of a potential double-up is so irresistible for some players on these tables, that hands like AQ to A9, tens, sometimes even nines, or KQ call an all-in move even when the blinds are still small. Many times we might get a reward, which is exactly what we wanted. To make a pot of this size with a raise we would need a lot of drawers to join the pot. However, the joined strength of all those hands, even really bad hands, usually beats our hand. If there is nobody to call the all-in move, we just grab the pot as it is and wait for the next opportunity.

All-in: Our hand is strong enough compared to the odds offered by the pot and the chance of taking the pot without any resistance, or our hand is ultimate and we want to be called.

5.2.6.2 Particular zone strategies

Our zone system does not precisely match the Harrington's one. We have taken into account the hand sets which are crucial for the algorithm implementation.

The green zone Our green zone strategy is used when having M of more than 12. Here we have enough time to wait for a good hand. On turbo tournaments, this

is a zone where we usually stay for a while after the tournament starts. Basically, there two ways how to get out of this zone. The first is by increasing blinds - this is the usual case. It doesn't matter how good the player is, on a turbo tournament the blinds are rising so fast that even a professional player cannot keep his stack up with the blinds. The second way is less pleasant - loosing a significant part of the stack in a conflict, usually when an all-in move is lost. Once we drop from this zone there is usually no point of return. Therefore, we want to put our money only in the best possible hands. The other approach would be to try to take a cheap look at the flop with a hand that could potentially win a lot of chips.

As we mentioned before, in small tournaments it is a very good strategy to go directly all-in without any raising in case we have a super strong hand. Unless we bet enough we could get called by a lot of drawing hands. Betting a significant amount of our stack when the blinds are still small just tells others that we have a monster hand. Nobody except of another monster hand is going to play with us, except for players who would not hesitate to call any bet. However, with the all-in move the situation is different. At the beginning of a tournament, there are often players lacking deeper knowledge of the game who base their game-play on luck and gambling. They know nothing about their long run efficiency and base their play on a feeling that this time their hand is going to catch the nuts. Let us put a big emphasis on the "at the beginning of the tournament" part. These people won't last long and if we get a good hand, we should try to double up despite the fact that we might not get called. It is worth the "risk".

In addition, by this approach we are building a reputation of a super tight player, from which we could make the best of later. We could also be treated as an "all-in player" if we get lucky and pick up a few great hands in a short time. This, however, still works in our favor. We might even get called by better players with weaker hands who consider us weak and try to get rid of us as an easy target. Later, when we start to be more aggressive, we could probably afford to do so - even with bad reputation - from the position of a big stack.

The strategy for the algorithm and the green zone is therefore relatively easy. If we get a premium hand form the ABest set,; no more decision making is required and we go all-in. The queens and jacks are a delicate problem. At higher tables they must be played more cautiously, at the lowest tables they may still be considered a premium hand and treated the same way. Although the long run profit will be far behind the one of kings', it should still be a positive number at the small stake tournaments. Playing the queens and jacks in any other way might be very hard to code and the revenue could only be slightly better.

Very good hands to play with in the green are included in the APlus set. These are, however, hands that need to improve. If they hit the flop, they usually create a top pair with the best or almost the best kicker and the same applies to a flush or

a straight. These hands might be used for opening the pot or for calling somebody who put in a significant raise or went all-in. However, if more betting occurred, for example there was a huge raise and somebody went all-in, it is highly probable that at least one of them has a pocket pair. In this case these hands are not good enough. When facing one of the premium hands we are a huge underdog, against a pair we have a little less than 50%. Actually, there are not many hands left that re-raised a huge raise and could be beaten by one of the APlus hands. Maybe the ace-jack or ace-queen having the flush advantage or a slightly better hand, but that is all. Sharing the same hand with one of our opponents is not good either. If we win we need to split the prize.

There are no more hands that are worth going all-in on pre-flop in the green zone. However, hands like tens, nines, ace-jack, ace-ten suited or suited connectors to queen are worth a standard play, which is a raise and then see what comes on the flop. We do not need to afraid to call a smaller raise with these hands, even if multiple people called that raise. In this case we are not going to re-raise of course, the primary goal to chase away the limpers was already done for us.

Another idea on how to make some chips in the green zone is to try to see the flop with a pocket pair. Of course, try to see it as cheaply as possible and let the hand go if we do not hit a set. If the flop goes well, we have a very good position that is hard to read, for almost any type of action. However, this play works only with very small blinds and should be abandoned when the potential of winning significant amount of money is outbalanced by the call size and the probability of hitting a set on the flop or later without additional costs.

The middle suited connectors can be played the same way as the pocket pairs. But in this case it is a little bit harder to pass judgment on where our hand stands. We have to consider the possibility of higher straights or flushes and be able to play accordingly, which is a harder task compared to the “set hunting”, especially for the algorithm. With these hands it is more likely to fall into a trap if we hit a good hand, but not the best one around a table. It is questionable whether we should try to play these hands in the green zone and whether the possible benefits are worth the risk we need to undergo. The algorithm is much more confident in situations where it is obvious where the hand stands.

The blinds in the green zone are usually so small that it is nonsense to try any pot stealing from the late positions. The revenue is simply too small compared to possible risks. However, calling the big blind from the small blind position with any ranked hand in our sets is usually worth. It costs only one half of the big blind to see the flop. Unless the big-blind raises of course.

If we get on the flop from the big-blind position for free, or from any other position when hunting for a hand, we should be cautious rather than aggressive. Even the top pair is not enough to risk our chips, especially when we were raising

with an A hand or a drawing hand. Two pairs might be the minimum for any betting or calling a bet. We might get outplayed by a worse hand in many cases, but again, there is enough time to wait for a better situation with a better hand. In case when there is only one opponent left and the flop is favorable we might try a bluff and raise with any hand. If we are acting from later positions he might think that we hit the small cards. If we are acting from the position of a limper or a raiser the big cards on flop might look scary.

The yellow zone With M still above 8, but no more in the green zone we need to shift our strategy to a more aggressive style a little bit, when compared to the previous zone. More and more hands become useful for some action as the time for hitting a great hand is becoming more limited. On the other hand, some drawing strategies that worked fine in the green zone might reach the breaking point and should be abandoned. We should be more aggressive especially in the late positions with the BPlus hands when nobody raised before. By taking bets from limpers and the blinds from time to time we will slow down the transition to the next zone. The limpers will usually fold, but we might get called by one of the players acting after us if he gets a good hand. But still, even with the BPlus hands we have a lot of outs in the situation.

If our hand is not good enough for an all-in move we may consider the blind steal with the B hands if the situation on the table allows it. This means no raisers or limpers are present and we are in a late, preferably the dealer position.

The orange zone Once again, for M above 4 there is a shift in the play to even more aggressive style. At this point many hands that were folded in the previous zones are becoming playable. Usually we can not afford to just call something and see what comes on the flop, because we are putting at risk a significant part of our stack. All-in moves should be now considered in many cases. Here, in the later positions, even the small pairs become applicable again. But this time, however, for an-all in move. With some chips left we still have some respect and a chance to pick up the pot right away. If this is not the case, then we will most often run into two over cards where we are a slight favorite or into another pair. We should definitely try to steal the blinds from the late positions with the B hands, from the dealer position even with the D hands.

The red zone With M less than 4 there is no maneuvering space left at all. There are only two actions possible, fold and all-in. All hands with a glimpse of thinking about an action in previous zones become premium hands in the red zone. The basic information we need for making a decision is whether anybody already entered the pot. If not and there are not many players left to act than an all-in move should

be done with the B and D hands. When the M drops even more we should include the C hands and at the edge with the dead zone the E hands as well. With the red premium hands we should not hesitate to set on a caller or raiser. Doing so against multiple re-raise in front of us should be reconsidered and proceeded with only with hands that have a real chance to make it, which are the C hands and the BPlus and better.

There is no need to handle the dead zone in a special way. Any ranked hand is good enough for the last call. Maybe on the bubble when it is possible that somebody else might die earlier. This is when there is a player in the same bad shape as we are or when there is a player all-in and is called by another player. This is the right place to fold almost everything, maybe the A and better hands should be played if there is a chance to get back to the game and play for a better position than the first paid.

5.2.7 The play

We are able to see the whole system up and running independently for the first time. Although it might seem that we have implemented only a fraction of the algorithm that can not work without the real poker math, the opposite is true. We have a solid strategy that can catch up during the whole tournament and adjust the game play accordingly. Of course, we can not expect that this strategy is good enough to permanently make profit in a real casino, but we can say that we have successfully merged with people around a table as an average player.

At this point, another phase of algorithm development should be started. An iterative process of playing, testing and adjusting. As we discover the cases with wrong or unsatisfactory play, we can adjust the algorithm decision methods, the hand sets or the M intervals. Later, we will probably need to add more strategies for specific tournament phases, especially for the bubble, the heads up, playing as a chip leader etc. We have to take into consideration more and more factors to be able to take advantage of these situations. These factors are forcing others from the position of a chip leader, or from the position of higher stack on the bubble, or play tighter on the bubble when there is somebody going to be eliminated. Many more methods can be found in poker books. It is up to the reader, or the implementer of an algorithm, to judge the appropriate benefits and drawbacks to the algorithm and the burdensomeness of corresponding implementation.

We have spent a while with this iterative tuning and debugging of our algorithm to be able to play during the whole tournament without any fatal mistakes. This will allow us to measure the algorithm efficiency against the Poker Academy bots. We have also tried to play according to this strategy in a real casino to see whether it has a potential to work.

There is a difference in whether we are testing our strategy in the Poker Academy

Pro or in a real casino with real people. The robots in the Poker Academy are trying to play mathematically correct poker. The Sklansky's all-in theory does not work here as good as in a real casino with humans. The value ascribed to our hand by their robots is often higher when we go all-in instead of a significant raise and we pick up some pots that we would not pick up for free in the same situation in a real casino. On the other hand, we are often correctly called in the Academy if the pot odds are good enough for the opponent. In the casinos some people are afraid more and give us free chips by their wrong play in these situations. Another huge difference between the Academy and a real casino are the observations. It seems that the bots do not take into account what the other players are doing. An occasional all-in move of another player has the same strength for them as a third all-in in a row. This is something that is noticed very well on the real tables and the humans become angry and start calling with weaker hands to protect their blinds and stacks.

Another very interesting observation that can be made is the overall strength of the algorithm against robots in the Academy and against humans in the real tournaments. It seems that playing according to our algorithm is more efficient in a real casino than in the Poker Academy. However, to be able to make a trustworthy conclusion about this, we would need to play more tournaments on both sides. This is not possible due to the limited demo version of the Poker Academy Pro we are using and the inability to let the algorithm play automatically in a real casino. Our claim is therefore only based on personal feeling gained by the results of a few trials. This is not that much surprising, considering that we have designed the algorithm to compete against humans knowing their fragility and eminency.

In this phase of our no-limit tournament algorithm implementation we have decided to stop our development. The reason was that the algorithm started to be better than we have expected it to be. We have not yet implemented and included the bubble play, tournament prize structure, low M vs high M confrontations[2] and other essential concepts including detailed probability analysis for particular situations[2].

5.3 Results

We were able to play 4281 hands in 72 tournaments in the demo version of Poker Academy Pro. We were using the same tournament setting as described in chapter 5.1. Let us remind this was a ten player turbo tournament with entry fee of $5.5 with $0.5 for the casino and a $50prize pool. The winner takes $25, the second $15 and the third $10. The complete list of these tournaments with every action and every hand is included in the addendum[3]. The number of games we were able to play is still not high enough to make any clear evidence and to completely eliminate

[2]For details see the devoted section 5.4.

[3]More information about game log can be found in addendum A.

the luck factor, however some consequences and results for our thesis' objectives can be made. We have faced 56 artificial opponents from the Poker Academy Pro during these tournaments, but nearly half of them were playing with us less than three or four tournaments. Therefore, we have decided to present the results only for the opponents with more than 1000 hands played.

One very important result that we have achieved is that during these hours of testing our system stayed up and running all the time. We have not experienced any crashing or freezing. Likewise, we have not noticed any redraw based issue that we were worried about at the beginning, when we were designing the data extraction and recognition system. This proves that it is possible to build a Self-acting Internet Casino Entity based on screen recognition and emulation of peripherals.

5.3.1 ROI

A poker player tournament profit is measured in return of investment in percents. (ROI) This is a ratio between the overall profit and all tournament fees and rakes paid. In the table 5.3 we have listed the overall results of every player with more than 1000 hands played. The most important value for making any conclusions is the ROI. The player names match the bots' names from the Poker Academy. We were playing under the nickname "Sice".

Player	Profit	Rake Paid	Hands Played	Games Played	Average ROI
Al Inne	-$42,50	$12,50	1582	25	-30,9
Amanda	-$43,00	$13,00	1496	26	-30,1
Angus	$3,50	$11,50	1631	23	2,8
Daniel Xn	-$51,00	$11,00	1158	22	-42,1
Evil Lynn	$2,00	$13,00	1858	26	1,4
Guru	$29,00	$11,00	1723	22	24,0
Gus Xensen	-$4,00	$9,00	1014	18	-4,0
Harmony	-$15,50	$10,50	1447	21	-13,4
Jack Pott	$12,50	$12,50	1585	25	9,1
Max Bette	-$61,50	$11,50	1621	23	-48,6
Mazama	-$26,00	$11,00	1559	22	-21,5
Odin	$48,50	$11,50	1890	23	38,3
Olea	$1,00	$9,00	1347	18	1,0
Oliver Chips	-$23,00	$8,00	1091	16	-26,1
Phil Tilte	-$45,00	$10,00	1335	20	-40,9
Pondera	-$17,50	$12,50	2027	25	-12,7
Ray Zure	$4,50	$10,50	1394	21	3,9
Sergio	$13,50	$11,50	1761	23	10,7
Sice	**$89,00**	**$36,00**	**4281**	**72**	**22,5**
Strongbad	-$65,00	$10,00	1189	20	-59,1
Zeno	-$4,50	$14,50	2316	29	-2,8

Table 5.3: Bot results and statistics

The bigger dispersion of ROI of other player is caused by a lower number of games they have played. Our ROI in our case has much better predicative value than the ROI of other players because it is based on three times more cases. Among the real players it is hard to find a person with ROI of more than 25% considering the player has played a representative number of games. The nature of Texas hold 'em poker creates some limits which are given by the quality of the opponents. A human player with a high ROI probably starts playing on higher-stake tables at some point in time. His ROI will drop down as he faces better opponents, but his income per hour will probably increase. In our simulation this aspect was not considered. If the results reflect bot qualities then Odin, Guru and Sice would probably start playing on higher-stake tables and it would be appropriate to measure the profit per hour, which is a more important indicator for somebody operating a poker bot.

Now it is time to answer another question of this thesis and make a conclusion on whether it is possible to build an intelligence capable of playing no-limit Texas hold 'em tournaments. Judging by the results we have achieved with the Poker Academy we could say it is. The reality is however a little bit different, our opponents were only other bots and we know nothing about their ability to perform against real people. How plausibly can the Poker Academy emulate an atmosphere of a real poker tournament is a matter of question too. At this point we can only confirm that we do not need a large development team working on the project for several years to create a robot which can compete with the best known no-limit tournament bots available. Another question is whether the Poker Academy team published the best bots they had or whether they kept the best no-limit tournament bots for the real casinos. Although, according to their results in fixed-limit ring games this is very unlikely.

We have also tried to play according to our strategy and see how it stands in real casinos. Regrettably, in the final phases of the algorithm development this was almost impossible due to many random decisions and spreading different strategies in our algorithm. We simply can not predict the actions good enough. There were no exact data created, but the play we were trying was rather successful than loss-making. At least on the very low-stake tournaments with different payout structure that we have determined to solve. When playing our desired tournament type we were not taking advantage of many situations around the bubble phases or in cases where we were the chip leader. We have not yet implemented these strategies in our algorithm. It is highly probable that we would be able to break even with a little bit more algorithm development even on our desired tournament type. We were usually trying our strategies during player attendance peaks in the casinos, which is also a time when there are many unexperienced players at the tables and winning is a little bit easier. On the other hand, these periods are the most suitable for money-making using robots. We can consider our results as a success in achieving the goals of this

thesis.

5.3.2 Overall profit

After seeing the graph of our money progress during all 72 tournaments, we were not very delighted. We knew the most important value, which is the difference between initial and final account balance from the previous chapter. However, the influence of a short-time luck factor is significant. In Figure 5.1 our bankroll size after each tournament is depicted.

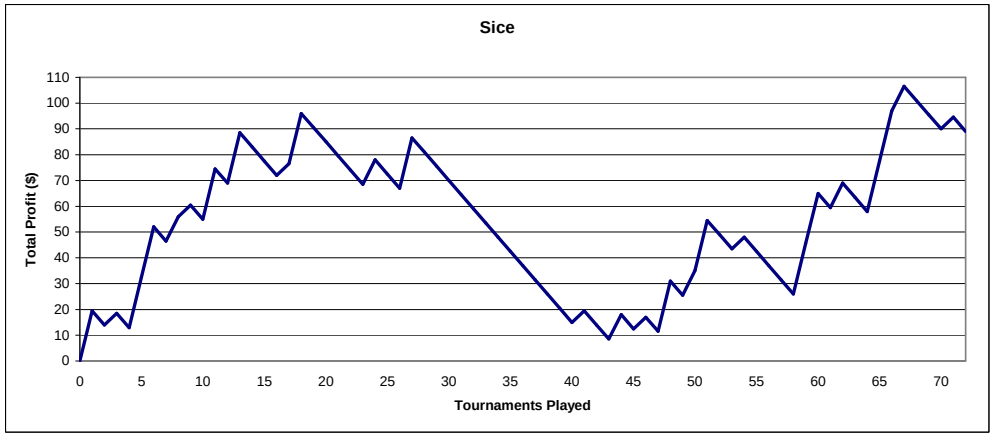


Figure 5.1: Overall profit

We have therefore decided to make additional measurements to confirm credibility of these results. We are going to challenge the best bots like Guru, Jack Pott, Odin and Sergio in a "bot-stars" competition right after we are finished with these results.

5.3.3 Places

We have achieved a very good structure of positions where our bot was eliminated. The Table 5.4 is depicting number of eliminations for each place in the tournaments.

Place	Times finished
1	13
2	6
3	7
4	7
5	7
6	6
7	7
8	5
9	7
10	7

Table 5.4: Places

Our algorithm is designed to win. This is the only place it is interested in. It does not care that being second or third is sometimes better than drop out fourth or fifth in the run for the first place. We are going to discuss some methods how to improve this in the chapter 5.4. Another thing that should be noticed is the perfect balance of the places where we have been eliminated. This means that with help of Mr. Sklansky and Mr. Harrington we were able to create a perfectly balanced strategy. The shifts in aggression based on the M-ratio and the hand sets are accurate and respecting every phase of a tournament.

It also shows that we are not afraid of loosing our chips and when we get a good hand we are aggressive with it. Otherwise, our drop-out positions would be concentrated in the middle. Being very tight from the beginning can reduce the probability of our elimination in the early phases of a tournament. On the other hand, it also means that in the middle of the tournament we would have lack of chips and would need to gamble with almost every hand to survive or be blinded away. The opposite strategy would be to play more loose from the beginning. The distribution of the final places in this case would be the exact opposite. We would be wiped out very often at the beginnings of the tournaments, but if we get through our chip stack would be big enough to make it to the finals. In this case it does not matter whether we will be playing more loose. We could afford to do that from the position of a big stack or a chip leader.

This also adds a trustworthiness to our measured ROI values from the previous sub-chapter. It flattens the differences between our more lucky seasons at the beginning an at the end of these measurements and the bad luck we had in the middle.

5.3.4 The bot-stars challenge

To support our previous measurements we have decided to run some more tests. We have used the same game settings as before. Only the opponents have changed. We have selected ten bots that made the best ROI during our experiences with Poker Academy and have played a non-trivial number of tournaments. Luckily, our bot

made it to this ten. This time there will be no players who give their chips away easily around the table, everyone who wants some extra chips must take them from the best. The finalist are the profitable bots from our previous measurements which are Angus, Guru, Jack Pott, Odin, Ray Zure, Sergio plus some other bots we have noticed during the testing which had a very good ROI. These are Chuck Reese, Kale and Xengrenau. The whole game-play history can be found in addendum[4]. We have to annul the tourney 14 with four players still in the game (Angus, Odin, Sergio, Sice) due to a Poker Academy crash. This tournament was removed from the game log and all entry fees were refunded. We were interested in the same values as before, this time we have created stats for everyone from out bot-stars team. Table 5.3 shows the overall game statistics.

Player	Profit	Rake Paid	Hands Played	Games Played	Average ROI
Angus	-$10,00	$15,00	1963	30	-6,1
Chuck Reese	-$45,00	$15,00	2097	30	-27,3
Guru	$45,00	$15,00	2278	30	27,3
Jack Pott	-$50,00	$15,00	1962	30	-30,3
Kale	$40,00	$15,00	2111	30	24,2
Odin	-$75,00	$15,00	1632	30	-45,5
Ray Zure	-$45,00	$15,00	2240	30	-27,3
Sergio	-$30,00	$15,00	1745	30	-18,2
Sice	**$45,00**	**$15,00**	**1838**	**30**	**27,3**
Xengrenau	-$25,00	$15,00	1809	30	-15,2

Table 5.5: Bot-stars results and statistics

This time the ROI dispersions are much tighter than before. One thing is, however, an outstanding surprise. It is the huge failure of the biggest favorite, Odin. A sovereign winner of the first competition and an absolute looser in the bot-stars competition. We have investigated the game log to figure out what happened. It shows up that Odin is not as a good player as it seemed. It is capable of calling against the odds, raising with nothing, going on a lot of draws and slow-playing good hands. The question therefore is: How could Odin make so good results in the first case? We have started an investigation there. It turned out that the main difference were the opponents. Odin does the same, plays any suited combination, calls everything with both cards above ten and plays with connected cards. The opponents often let him draw for free and later, if he hits something, they give him a significant portion of their chips assuming he has nothing. It seems that Odin was designed to profit on the worst players and to teach users of the Poker Academy how important is to bet with good hands. The same applies to his wrong slow-plays that worked good against the worst bots. In the finals it has no problem to outplay

[4]More information about bot-stars game-log can be found in addendum A

other stupid bots with his wrong raises. When a bot does not understand very well the need to steal and protect the blinds at a tournaments it is not a deal to outplay him. The bubble play is something that Odin is not very familiar with as well. This is, however, an important lesson for a poker bot designer.

- A given poker bot strategy is as good as the contra strategies of his opponents are bad.

In real casinos, the range of player strategies is much wider when compared to bots in simulated environment. It shows that it is important to focus the poker bot strategy development on the target opponent type. What might work perfectly with some opponents does not necessarily work with other type of opponents. However, a good balanced strategy should work everywhere. Both, Guru and Sice did well with both type of opponents. The results we have achieved are far beyond everything we have expected. This could be definitely considered as an accomplishment of our goal that we have set out for the Poker Academy Pro and our bot. We stayed up and running all the time and even among the best bots we were able to reach the top results.

However, the real winner of this competition is the casino. We have expected and predicted this earlier. The profit of the casino is three times bigger than the profit of the best bots. This is also the reason why most of the bots are in a loss. If there were no rakes for the casino Angus and Xegrenau would have achieved better results, at least closer to the edge.

Let us take a look on overall player profits during the whole competition. This time we have created a graph for each player. The results are depicted in graphs in Figure 5.2.

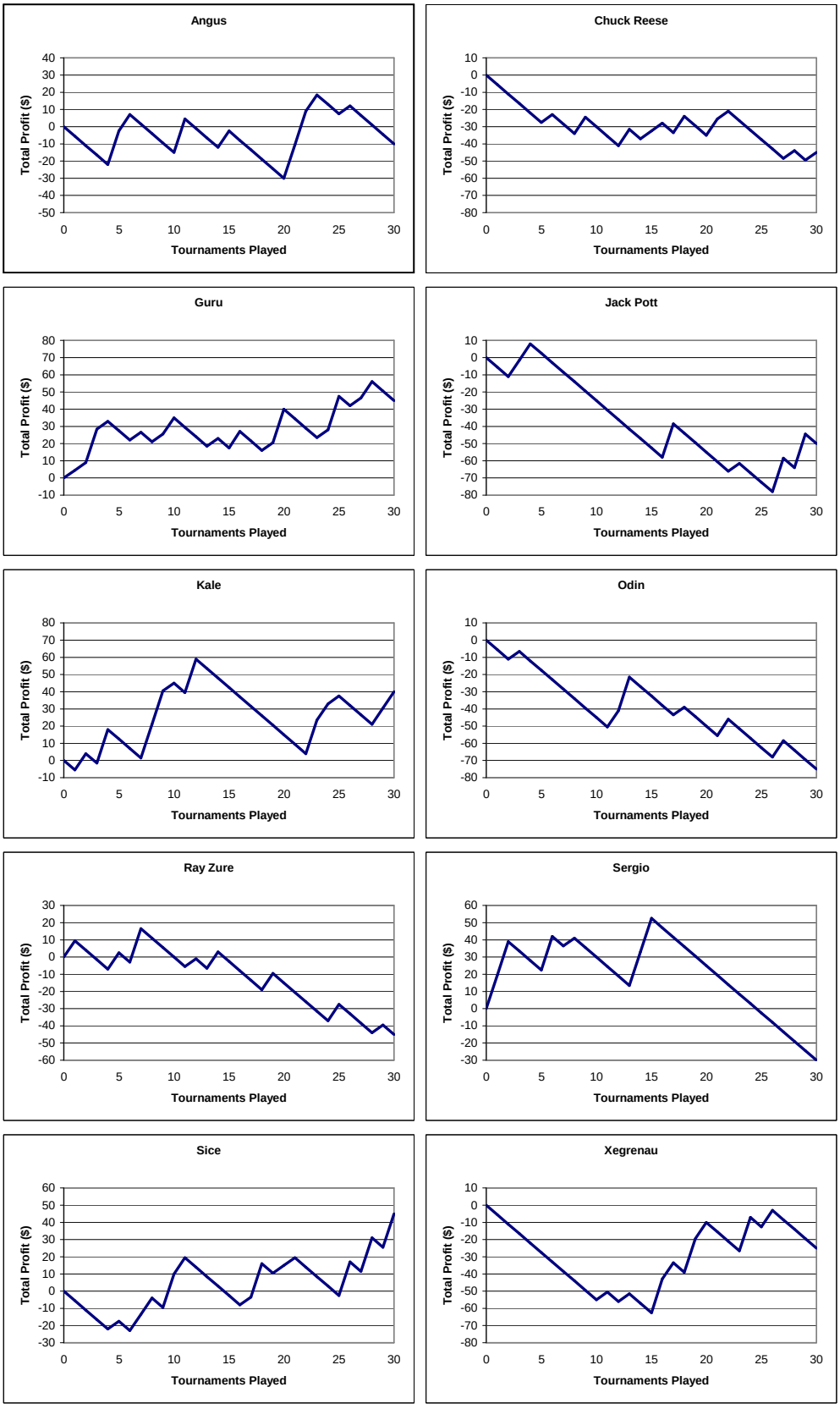


Figure 5.2: Bot-stars overall profit

As we see, in most of the cases the behavior of the curves has a clear slope. From this point, the 30 measurements that we were able to do in the demo version of Poker Academy Pro seem sufficient, but it is at the edge. For Sergio and Xengrenau these limits are fatal, we can not able to deduce any conclusions and more measurements should be performed.

We have also created a table with numbers of positions where each player was eliminated. These values are depicted in Table 5.6. We will add some comments on these results that may be helpful for the bot tuning process.

Place	Agnus	Chuck R.	Guru	Jack Pott	Kale	Odin	Ray Zure	Sergio	Sice	Xengren.
1	4	0	3	3	5	1	1	5	**5**	3
2	3	4	3	2	4	3	5	0	**3**	3
3	1	6	9	1	2	2	2	1	**4**	2
4	2	2	0	4	2	5	6	3	**2**	4
5	3	4	3	4	4	3	3	2	**1**	3
6	5	5	3	3	2	4	3	1	**4**	0
7	2	6	1	2	4	1	2	4	**4**	4
8	6	2	2	3	1	4	4	3	**1**	4
9	4	1	2	3	2	1	3	6	**5**	3
10	0	0	4	5	4	6	1	5	**1**	4

Table 5.6: Bot-stars places

This time there were no donors at all. Every bot we have faced made it to the finals more than once or twice. The competition was really though. We can learn something from the results in this table.

Chuck Ray is a very tight player. Here we can see a situation which we have described earlier. It is trying to stay in the tournament for higher costs than he should. It has too little drop outs from the beginning of the tournaments. But he had lack of chips later and was eliminated very often in the middle stages and as the only bot he never won.

Sergio seems to be the exact opposite. It does not mind to undertake any risk from the beginning to win the tournament. However, the risk he is taking is probably too high and therefore he failed to be profitable overall.

Guru is probably the best player of all, or at least he has the biggest potential. On the other hand, he completely failed on the bubble. It was trying to survive it to the final three at any costs. Guru succeeded, but the prize it paid was very high. Afterwards it was very often eliminated at the third place. Guru should adapt one of the methods which take into account tournament prize structure and act accordingly. Sometimes it is better to risk and finish fourth rather than failing to win later.

Sice has once again a well balanced overall drop out structure. However, this time we had to face better opponents which means it was harder to win that many times as before.

5.4 Further algorithm improvements

Despite the fact that there is still a lot of space for further improvement of the algorithm in the area of states and basic strategies, there are some advanced strategies whose implementation could significantly improve the algorithm game play. Up to this point, we have designed the strategy only according to already publicized guides on how to play given hands in specific situations. These guidelines of course respect the statistical probability for each hand, but can not take into account the specific conditions on the table. Here we would like to introduce some advanced approaches that are based on probability analysis of given situations. Originally, we thought that creating a poker AI is impossible without implementing a lot of more-sophisticated methods and that the algorithm would contain a big number of combinatorial computations. We did not suspect that a balanced combination of Sklansky's all-in theory, Harrington's zone system and our state automaton approach to solve the post-flop situations could be so powerful on its own. By implementing these techniques we would probably improve our poker AI's strength dramatically. We have only picked up a few ideas that can significantly improve our play mostly in the middle and late stages, as we know this is the Achilles' heel of our algorithm at this point.

5.4.1 Fold equity

Fold equity[5] is a crucial concept to understand in order to be able to win no-limit tournaments. In the later tournament phases, especially on the bubble, when the blinds cut a significant portion of players' stacks every time, the role of the cards is not so important anymore. Simply said, if there is a decent chance to take the pot containing blinds uncontested, we should try to do it. However, we have to compare the potential profit with the risk of loosing a significant portion of the stack, as this can happen in many situations. Fold equity is the difference in gain between a fold and a raise when trying to pick up the pot. It can be either positive or negative and characterizes expected gain of the "stealing" move. However, to compute the fold equity we need to employ opponent reading skills to be able to estimate the likelihood of them folding to our bet. If we could do so, then we just need to subtract the chances that everyone folds times the pot from the chances that we get called times our raise. The result is the fold equity of the raise move. Note that the higher the raise is, the better chances that everyone folds we get, but the bigger portion of our stack is being risked on the other side.

Now, a question of how to make estimations of our opponent's play arises. We need to incorporate some observation techniques. In our environment, this requires additional implementation in the used front-ends as well. We need to decide which facts we are interested in for our algorithm development and use the additional table

data construct to provide these information to the back-end. In this concrete case we are concerned in the likeliness of a player calling or raising a raise. This value can be easily computed iteratively by trying to raise our opponent and seeing how many times we get an active response. However, to make this value reliable a few iterations are required and it is questionable whether we can afford to make them. We should try to estimate these values during the tournament so we can rely on them from the first moment. This can be done by observing how likely a player is to play post-flop, how big the pot is compared to the blinds when the player is involved, etc. This way we can gain a solid notion of each player's game style, information on whether he is loose or rather tight and whether he is likely to fold or to act.

Another way of making these estimations more accurate is to use an external player database during the play, for example the SharkScope[5]. If we are facing a profitable player we can be pretty sure he understands what is going on and does the corresponding contra action. On the other hand, we can be facing a weak player. There are generally two types of weak players. The first ones play very tight and are bluffed easily, the second ones play very loose and make a lot of wrong calls. Luckily for us, we do not need to distinguish between these two categories that much, because the same tactics can be used to deal with both player types. We should bluff the tight player with every at least a little bit reasonable hand. If we get called, we are an underdog almost every time. Therefore, we should prefer the drawing hands that can win from time to time to the hands that are very unlikely to win the pot. The same, however, works on a loose player. We know that he is capable of calling with worse than average hands and this fits to our strategy too. If he folds we get the pot for free and if we get called his hand usually won't be much better, as he is a loose player. This approach is, however, against most of the casino agreement rules, but could be a strong weapon for somebody operating an automated player for profit.

5.4.2 Structured hand analysis

An expansion of the fold equity idea can be made to cover the middle stages of a tournament. The structured hand analysis[2] can be used when there are no players in the red zone, but the blinds are relatively high. As the reader might have noticed, we have not included the card factor in the previous idea very much, especially the cards that the opponent might have for a contra action.

This does not matter from the algorithm point of view as the raise would be often an all-in move. In this phases of a tournament there is usually not enough space left for trying a bet and subsequently folding it if we are re-raised. The pot odds are often simply too good to fold. Even if there was space for such maneuvers,

[5]More information can be found at *http://www.sharkscope.com/*.

we can still apply an all-in steal. It is easier to implement and the algorithm cannot be outplayed, like the bots in Poker Academy, with a right re-raise. It also increases the chances that we will not face active resistance to the maximum.

This approach is based on situation analysis from the other player's point of view. Once again, we can not leave out players' play style estimations. We will use the same method as described earlier in this chapter. According to these estimations, we are able to create a list of hands that each player will use for a contra action. The supplement is a list of hands that the opponent folds. We will make a slightly inaccurate assumption that only one person could call us. This assumption will allow us to avoid getting lost in all possible combinations of calling players in the following explanation. Moreover, if we go all-in and get called, there are not many hands that the third player could afford to call a double all-in with. We have estimated a list of hands that will call our all-in move for each player, based on his play style.

The aims of the second phase are to calculate probabilities of us winning or loosing the pot when we get called and calculate probability of taking the pot without any resistance. Based on the hand lists from the first phase we can compute probabilities of loosing or winning against each opponent with our hand. Furthermore, we compute overall probabilities of getting called by each player[6] and of winning the pot uncontested.

In the third phase we compute an overall expectation of the all-in move. The difference in stack sizes between our opponent and us tells us how much we can win[7] or loose in case we get called. Knowing the probabilities from the second phase, which are how likely we get called and win/loose afterwards, we can compute the expected gain against each player[8]. A sum through these gains for each player to act plus the income if we win the pot uncontested[9] is the expectation of the all-in move. After comparing the expectation of the all-in move to the "expectation of the fold move" we see the decision we should make.

We have mentioned the structured hand analysis because the computer, as opposed to a human, is able too use it during the play. On the other hand, the human can probably estimate the action hand list more precisely. This assumption is also the weakest point for the algorithm. We can simplify these assumptions by creating one global hand list sorted according to pre-flop hand strengths. Afterwards, for each player we can try to find a projection from his observed style to a position in this sorted list. Another approach would be to put ourselves in the position of our opponent and ask what actions we would take for each hand from the list. In this

[6]This is a ratio between the number of ways how to deal all hands on our opponent's hand list and the number of all ways how to deal two cards.

[7]In case that we win we have to include the actual pot to our favor.

[8]This gain is usually negative.

[9]This is the value that should compensate the negative gains we get against our opponents.

case we can also adjust the M to reflect the corresponding player's style. This means we increase our stack for tight players and decrease the stack for loose players.

Let us note that this model also works the other way around, when we are trying to decide whether to call an all-in or not.

5.4.3 Independent chip model

Once again a careful reader might have noticed that in both previous methods we were talking about a profitable move in terms of increasing our chip stack. This is true for the ring games, but the term profitable should have a different meaning in the tournament play. In a ring game, the profit equals to the increase of chip stack. In a tournament, the profit depends on the payout structure. The independent chip model[6] takes this into consideration and extends the structured hand analysis even more. This model is an ultimate weapon for the algorithm and allows us to make correct decisions from middle phases to the heads-up.

There are many examples and notes available in the literature, we will therefore introduce the concept only briefly. The model is based on two simple assumptions.

- A player's chances of winning the tournament equal his share of the total chips[4].
- Every player on the table has the same poker skills.

Even though the second assumption is not correct, the calculations and results of this model are still good enough and applicable. The independent chip model computes a real value of each chip of each player according to the tournament prize structure. This is based on the first assumption that allows us to compute probabilities for each player finishing on each remaining position in the tournament. Knowing the tournament prize structure, we can count the real value of each stack. This value is called the *icm equity*. When we face a decision, it does not matter whether we go-all in or call an all-in, we can use the structured hand analysis the same way, but instead of computing an average expected profit we will use this chip model to compute the real value of our resulting stack. After comparing the icm equities we see which situation gives us a higher value of our stack in the resulting situation.

This was short explanation of the model, that should give the reader a notion what it is about. Note that a profitable play in terms of size of our chip stack does not imply a better icm equity. Simply, we might be risking to drop out from the tournament with no or lower prize. Even if the chances of dropping out are small and in long run we win extra chips in such situations, the overall amount of money we win in tournaments might decrease. This of course applies in the opposite way as well. We might make a move which we know according to the structural hand analysis is incorrect, but we are increasing our icm equity. This might happen due to

a possible elimination of other player and the resulting increased chance of finishing on a better place in the tournament.

Chapter 6

Conclusion

We were not able to fulfill the goals we have stated at the beginning of this thesis to full extent. We had to slightly modify and adapt our goals to new conditions and possibilities as we encountered new facts and restrictions related to the casino usage policy and work already done in this area.

With regard to these adjustments and restrictions, we have succeeded in every objective designated in this thesis. Moreover, we were able to surpass the aims that we have determined for certain areas. From this position we are able to answer the target questions and summarize the crucial knowledge gained during the implementation and investigation process related to this master thesis.

6.1 Summary

In the early phases of our investigation we have found out that there already are poker robots available. Some of them are being developed in the academics, some of them are also intended for playing games at on-line casinos. Most of these robots are designed to play fixed-limit cash games as these are easier to compute. Further discussion of this topic can be found in chapter 2.2.3.3. At this point we knew that creation of at least a basic poker bot is definitely possible. We have therefore decided to specialize our investigation and opening questions to the no-limit sit and go tournaments and fulfill our robot related goals for them as the less explored hold 'em poker variation.

6.1.1 Environment for poker bot creation

The environment for automated poker bot creation we have developed and implemented in .NET and c# consist of two primary parts – a front-end and a back-end. The first one is responsible for gathering data and interacting with casino software,

the other provides the game logic and decision making. The environment is implemented as a set of modules and supportive tools that are used by a designer to create a library which handles given tasks. We wanted to provide the designer with the full strength of a programming language and avoid restricting him with any limitations of our environment as happened in many cases of the existing software. We have also created a GUI which makes it possible to manage the entire system.

6.1.2 Front-end

The whole process of gathering data from a casino is based on screen observation and optical recognition. This is the most universal solution that should work with majority of on-line casinos. We have built a WYSIWIG editor that allows the front-end designer to easily select the strategic visual components of a casino and related poker tables. These data are afterwards available through our modules with a lot of additional functionality. Subsequently the front-end just inquires the necessary data, sends them to the back-end for further processing and later executes the requested action. To execute actions in the environment of a casino we use peripherals emulation.

We have tested the functionality of our environment for front-end creation by creating an implementation of two front-ends, for the PokerStars and for the Poker Academy Pro.

For the PokerStars casino we have also implemented window management and automatic tournament registration. By using the PokerStars front-end we are potentially able to play on multiple tables at once and keep a constant number of running tournaments without any human interaction.

6.1.3 Back-end

A lot of work, including development of a method of programming algorithms for no-limit tournaments, has been done on the back-end. We have created a methodology and a supportive set of modules that allow a very effective no-limit tournament strategy implementation. The whole idea is based on the fact that for no-limit games it is much more important to keep a track of what happened in the previous betting rounds, compared to fixed-limit games. We have invented a state automaton approach where every user defined state represents a betting pattern from the previous betting rounds, thus allowing the user to make more competent decisions than with use of any other software based on simple rules for every hand.

We have tested the functionality of our environment for poker algorithm creation by implementing an exemplary no-limit tournament strategy. This strategy was based on a combination of Harrington's zone system and the M-ratio concept

described in Harrington on Hold'em - Volume II[2], Sklansky's hand sets with similar pre-flop strategies described in Hold'em Poker For Advanced Players[3] and our state automaton approach. This combination of approaches allows our algorithm to play all phases of a tournament on very high level.

6.2 Results

To answer our questions, we have tested our robot on Poker Academy Pro, which is a casino emulator for starting poker players developed by the games group of University of Alberta. This group is a leader in poker bot development nowadays, in July 2008 their robot has beaten some of the best fixed-limit cash players for the first time. Their robot is however only capable of playing against a single opponent.

We have used the combination of our respective front-end and back-end to demonstrate the possibility of automated play in this emulated casino without any human interaction. Our system based on image recognition was able to run for hours without any difficulties, starting a new tournament automatically right after the previous one finished. We were able to confront the skills of our bot with the skills of probably the best bots currently available.

Our results were very impressive. Not only was our robot able to make profit with the default bot structure which is trying to simulate a standard no-limit tournament in a real casino, it also tried to play against the nine best robots from the previous measurements. The results here were very encouraging. Our robot has finished as one of the three profitable bots out of the ten. We have showed that our simple idea with states combined with already known strategies can keep up with much more sophisticated approaches used in Poker Academy Pro.

Comments regarding our algorithm implementation are included as well. However, some parts of provided code were modified or removed to weaken the overall strength of the algorithm.

We have also noted that a bot strategy which works with one kind of players does not necessarily have to work with different strategies and players. A poker bot designer should therefore keep in mind the target opponents of his or her robot. Sometimes, even an imperfect play can deliver better results with given players than a strategy generally considered as superior one.

According to the measurements and the results, we have discussed possible weak and strong points of some bots. This might be helpful for tuning and improving the skills of one's owns bot.

At the end we have discussed some advanced and generally known techniques and strategies regarding the no-limit tournaments from the point of view of a bot designer. Implementation of these techniques in our algorithm may and probably will increase its strength significantly, far beyond the level we have imagined to be

possible to achieve when we started our research.

6.3 Our questions

It did not take a lot of time to figure out that creating an on-line poker robot is not only possible, but such robots have already been created. We have shown that poker robots are not predestined to play only at fixed-limit cash-game tables. A proof that no-limit tournaments are vulnerable as well has been created, although more knowledge and a more sophisticated approach was necessary.

We were not able to test our software directly on a real casino due to the casino's usage policy. However, based on a personal comparison of robot and player skills in the Poker Academy and a sample of real casinos, we can proclaim that it is possible to create a profitable bot playing no-limit sit and go tournaments as well.

Concerning the vulnerability of casinos, it is possible to operate a bot on every casino where a human is willing to play. By using our recognition system it is a question of one or at most two days to create a front-end that can be used to play on any casino with all the necessary visual elements that are needed for a tournament poker play. On the other hand, the casinos try to fight against the bots in many different ways. They cannot however do very much until the bot leaks to the public and is used more frequently.

We have shown that it is possible to create an operational and probably profitable bot without a huge team and many years of development. This answers the question whether there already are bots winning money from regular users, although it is impossible to estimate their number due to the necessity of keeping them secret.

The last constituting a problem is the poker bots' inability to lead a meaningful conversation in the chat. On the other hand, writing comments related to the actual game from time to time is not a problem for a robot. This could make the robot an unobtrusive member of the table, slowly but surely making money every night on any poker table in any on-line casino.

Bibliography

[1] Dan Harrington, Bill Robertie: *Harrington on Hold'em - Volume I: Strategic Play*; Two Plus Two, 2004

[2] Dan Harrington, Bill Robertie: *Harrington on Hold'em - Volume II: The Endgame*; Two Plus Two, 2005

[3] David Sklansky, Mason Malmuth: *Hold'em Poker For Advanced Players*; Two Plus Two, 2001

[4] David Sklansky: *Tournament Poker for Advanced Players*; Two Plus Two, 2002

[5] Lee Nelson, Tysen Streib and Kim Lee: *Kill Everyone*; Huntington Press, 2008

[6] Phil Shaw: *Secrets of Sit 'n' Gos: Winning Strategies for Single-table Poker Tournaments*; D&B Publishing, 2008

Appendix A

Attached DVD

Digital materials related to this thesis are included on the attached DVD. Please, do not make the DVD publicly available. The disc has the following structure:

- `data/` Data and logs related to the thesis.
  - `casinos/` Examples of casino's tables.
  - `gameplay/` Logs with hand histories from automatically played tournaments.
- `sice/` Environment for creating automated Texas Hold'em poker players.
  - `release/` Release version of the environment.
  - `source/` Source codes of the environment.
- `thesis/` Text of this thesis in PDF format.
- `video/` Exemplary videos in AVI format demonstrating Sice play, Trigger Editor usage and regression testing.

A.1 Sice environment structure

The `sice` folder contains the following directories and files:

- `bin/` Precompiled components of the environment. This includes SICE Activator, Regression Tester, SICE Trigger Editor.
- `data/` All data used by the environment including Recognizers and Triggers for provided casinos.

- `doc/` Program documentation of the environment.
- `source/` Environment for creating automated Texas Hold'em poker players (and source-code of the environment in case of the source version) for MS Visual Studio 2008 and MS Visual Studio 2005.
- `test/` Test scenarios for provided casinos.
- `README.txt/` Information about software usage and limitations, software configuration and supplied software modules.

Appendix B

Related data

B.1 Poker client observation

Table B.1 and Table B.2 show and overall information regarding visual and gameplay components in particular casinos.

Poker Server	Disable animation	Resizable window	Server specific table icon	Highlighted player to act	Const. card pos, color	Total pot	Blinds	Support for notes	Numeric fonts recognition	Card recognition	Player name recognition	The same position for texts
PokerStars	Yes	Yes	Yes	No (blink)	Yes	Yes	Yes (title bar)	Yes	Yes (text color changing)	Yes (antial., 4 color)	Yes	Yes
FullTilt	Yes	Yes	Yes	Yes	Yes	Yes	Yes (title bar)	Yes (opens in new window)	Yes	Yes (antial., 4 color)	Yes	Yes
Party Poker	Yes	Yes	No (whole window scheme)	Yes	No (win. c. – new pos & transp.)	Yes (antial. or current pot)	Yes	Yes	Yes	Yes (antial., 4 color)	Yes	Yes
Ultimate Bet	Yes	Yes	Yes	Yes	Yes	No (current pot only)	Yes (title bar)	Yes (opens in new window)	Yes	Yes (antial., 4 color)	Yes (overleaped with notes)	Yes
Absolute Poker	No	Yes	Yes	Yes	No (win. cards – new pos)	Yes	No	Yes (opens in new window)	No (human cannot read)	Yes (antial., 4 color)	No	Yes
Celeb Poker	No	No (supports minimode)	Yes	Yes	No (win. c. – new pos & transp.)	Yes	Yes (in popup menu)	Yes (opens in new window)	Yes	Yes (antial., 4 color)	No (scrolling)	Yes
William Hill	Yes	No (support small size	Yes	Yes	No (win. c. – new pos & transp.)	Yes	No	Yes	Yes (antial.)	Yes (antial., 4 color)	Yes (antial.)	Yes
Inter Poker	Yes	No (support small size	Yes	Yes	No (win. c. – new pos & transp.)	Yes	No	Yes	Yes (antial.)	Yes (antial., 4 color)	Yes (antial.)	Yes
Allstar	No	Yes	Yes	No (blink)	No (win. c. – new pos & transp.)	No (current pot only)	Yes (title bar)	Yes	Yes	No (antial., shadow overleap)	Yes	Yes
RedStar Poker	No	Yes	Yes	No (blink)	No (win. c. – new pos & transp.)	No (current pot only)	Yes (title bar)	Yes	Yes	Yes (antial., 4 color)	Yes	Yes
Titan Poker	Yes (not all animation disabled)	No (support small size	Yes	No (blink)	No (win. c. – new pos & transp.)	No (Yes in large window)	Yes (title bar)	Yes (opens in new window)	Yes	Yes (antial., 4 color)	Yes	Yes
CD Poker	Yes (not all animation disabled)	No (support small size	Yes	No (blink)	No (win. c. – new pos & transp.)	Yes	Yes (title bar)	Yes (opens in new window)	Yes	Yes (antial., 4 color)	Yes	Yes
Paradise Poker	No	No (supports minimode)	Yes	Yes	No (win. c. – new pos & transp.)	Yes	Yes (in popup menu)	Yes (opens in new window)	Yes	Yes (antial., 4 color)	No (scrolling)	Yes
Euro Poker	No	No (supports minimode)	Yes	Yes	Yes	Yes	No (no announce ment)	Yes (opens in new window)	Yes	Yes (antial., 4 color)	Yes	Yes
RedKings	No	No (supports minimode)	Yes	Yes	Yes	Yes	No (no announce ment)	Yes (opens in new window)	Yes	Yes (antial., 4 color)	Yes	Yes
Carbon Poker	Yes	Yes	Yes	Yes	No (win. c. – new pos & transp.)	No (current pot only)	Yes	Yes (opens in new window)	No (human cannot read)	antialiased, 4 color	No (antial.)	Yes
PokerCity	Yes	Yes	Yes	Yes	No (win. c. – new pos & transp.)	No (current pot only)	Yes	Yes (opens in new window)	No	Yes (antial., 4 color)	No (antial.)	Yes
Pacific Poker												
PokerHills	Yes	Yes	Yes	No (hidden for a while)	No (win. cards – new pos)	Yes	Partial (title bar + announce	Yes (unknown type of	Yes	Yes (antial., 4 color)	Yes	Yes
24h Poker	Yes	Yes	Yes	No (hidden for a while)	No (win. cards – new pos)	Yes	Partial (title bar + announce	Yes (unknown type of	No	Yes (antial., 4 color)	No (antial.)	Yes
Poker.com												

Table B.1: Graphical elements in poker clients

Poker Server	Playmoney tables	Table Filter	Supports notes	Turbo sit & go	Notes
PokerStars	Yes	Yes	Yes	Yes	
FullTilt	Yes	Yes	Yes (opens in new window)	Yes	
Party Poker	Yes	Yes	Yes	Yes	
Ultimate Bet	Yes	Yes	Yes (opens in new window)	Yes (+ultra)	
Absolute Poker	Yes	Partial	Yes (opens in new window)	Yes (+ultra)	
Celeb Poker	No	Partial	Yes (opens in new window)	Yes (+speed)	Email spammer
William Hill	No	Yes	Yes	Yes	
Inter Poker	No	Yes	Yes	Yes	
Allstar	Yes	No	Yes	Yes	
RedStar Poker	Yes	No	Yes	Yes	Email spammer
Titan Poker	Yes	Yes	Yes (opens in new window)	Yes	
CD Poker	Yes	Yes	Yes (opens in new window)	Yes	
Paradise Poker	No	Partial	Yes (opens in new window)	Yes (+speed)	
Euro Poker	Yes	Partial	Yes (opens in new window)	Yes	
RedKings	Yes	Partial	Yes (opens in new window)	Yes	
Carbon Poker	Yes	Yes	Yes (opens in new window)	Yes	Unstable, not working properly
PokerCity	Yes	Yes	Yes (opens in new window)	Yes	Unstable, not working properly
Pacific Poker					Unable to connect
PokerHills	Yes	Yes	Yes (unknown type of	Yes (+super speed)	
24h Poker	Yes	Yes	Yes (unknown type of	Yes	
Poker.com					Unstable, not working at all

Table B.2: Gameplay components in poker clients