Artificial intelligence and data mining applied to no-limit Texas Hold'em

Sjoerd Henstra & Robin van der Zwan

Leiden Institute of Advanced Computer Science
Universiteit Leiden, The Netherlands
shenstra@liacs.nl
rvdzwan@liacs.nl

January 30, 2007

Abstract

This project has two goals. First, to create a poker playing computer softbot for Sit and Go tournaments. This bot will be based on statistics and known strategies used by real players and should at least play better than a player new to the game. Second, to data mine records of poker hands played in the past and look for potentially interesting patterns that might be useful in creating a poker bot. The goal here is to at least find some interesting statistics about the way people play poker, and perhaps to even find a way to predict a person's cards by analysing his actions.

1 Introduction

Poker has a rich history of study in the academic field. Economists and mathematicians have applied a variety of analytical techniques to poker-related problems. In early investigations in game theory, people like John von Neumann [16] and John Nash [10], used a simplified version of poker to illustrate the fundamental principles.

Until a few years ago, the computer science community has largely ignored poker. The game has a number of topics that make it interesting to study. These attributes include incomplete knowledge, risk management, opponent modelling, multiple competing agents, deception, exploiting weaknesses by deducing playing styles of opponents, and dealing with unreliable information. And this is all combined in the game of poker [6]. Some of the work in game theory that was already done on poker could only handle extremely simple poker games. An example is Kuhn's game for two players, using a three-card deck, one-card hands, and one betting round, with at most two betting decisions [8]. This was enough

to demonstrate fundamental principles of game theory, but doesn't resemble the poker variations played today.

Recently the research on poker has exploded. Most of the work done on poker is on limit poker. This is very different from no-limit poker. In limit poker there is always a fixed amount you are allowed to bet but with no-limit poker you may bet any amount at any time. No-limit poker is strategically very different from limit poker. Because the whole stack is always in jeopardy, the risks and rewards are much greater in no-limit poker than in limit poker. To play at the level of the no-limit poker-pro's it requires a greater knowledge of the opponent, better judgement and more courage. In no-limit poker there is much more emphasis on the "human" characteristics. This is why most research on poker is on limit poker.

The University of Alberta Computer Poker Research Group has been researching Artificial Intelligence applied to Poker since the mid-90's. They have created a pokerbot called Poki which is considered to be one of the best computer playing pokerbots. Poki plays no limit poker. For more information visit their website [11].

When we started this project we had great plans to create a complete bot for playing No Limit Texas Hold'em at top level. However, during the creation it turned out to be an enormous project, so enormous we had to cut back on our plans. The new idea was to create a pokerbot who would win against basic (predictable) players. More about our opponents in the experiment section.

We initially planned to use a database of games of Texas Hold'em played in the past as training data for a neural network that classifies opponents. However, we quickly decided not to do this, but rather to mine the database for interesting information. Any useful information could be incorporated into the bot and tested. Due to time constraints this implementation and testing could not fully take place, but some of the results might be usable to create a stronger bot that can take on, or see through, certain traits of human poker playing.

This paper will start with a short introduction into poker and the type of poker that we are creating a soft-bot for. After that we have a chapter to discuss what exactly it is what we are trying to make. The rest of the paper is split up into three parts. The first part, which will be about the play of the bot and the calculations involved, covered in Sections 2 and Section 3, was mostly the work of Robin van der Zwan. The second part, which will be about data mining on logs of poker games in Section 4, was done mostly by Sjoerd Henstra. The third part will consist of experiments and results and we will finish with the conclusions. At the end of the paper is a glossary for specific poker terms.

This bachelor project was done under supervision of Walter Kosters from LIACS, Leiden.

2 Short introduction to Poker

There are many variants of Poker that are played. The research in this paper is applied to a specific variation of poker called No-Limit Texas Hold'em. No-Limit

Texas Hold'em is the poker variation used to determine the world champion each year at the World Series of Poker and it is one of the most popular versions of poker played in (online) casinos. No-Limit Texas Hold'em is typically played with two to ten players.

At the beginning of each hand of hold'em, the dealer starting at his left, and deals around until each player has two cards face down. The deal moves to the left after each hand. There is one round of betting on these first two cards. After the first betting round the dealer now deals three cards face up in the middle of the table. These cards are called the "flop." They are community cards used in common by every player still in the hand. There is then a second round of betting. After that, a fourth card is dealt face up and a third round of betting occurs. A fifth and final card is then turned up and a last round of betting takes place. If there is more than one player left in the game at the end, the winner of the pot is the player with the best poker hand using the best five out of seven cards (the two in his hand and the five in the middle). In each betting round a player can either fold (throw his hand away and lose possibility to win the pot), call (match what the opponent has raised) or raise (increase their investment in the pot). For a complete introduction to the rules of No Limit Hold 'em and some basic strategy we refer to [4] or [1], or other books on (No-Limit) Texas Hold'em.

There is no precise information on where or when Texas Hold 'em Poker was first played. According to legend, the earliest game played was in Robstown, Texas, in the early 1900s and it first came to Dallas, Texas in 1925. Texas hold 'em was introduced to Las Vegas by a group of Texan gamblers and card players, including Crandell Addington, Doyle Brunson, and Amarillo Slim. The game was later introduced to Europe by bookmakers Terry Rogers and Liam "The Gentleman" Flood [5].

No Limit Hold'em is not just a card game but a game of wagering based on imperfect information that uses cards and actions of opponents to construct the best action to take. To make a good estimate the following four factors have to be taken into account:

1. The likelihood that the hand will improve as the cards are dealt, which is straightforward mathematics.
2. An estimate of the hand that the opponent(s) may hold, which is based on the history of actions for each player.
3. The likelihood that the opponent's hand will improve, again straightforward mathematics, but complicated by the fact that the opponent's hand is not known for sure.
4. The money odds offered by the pot.

When a good player plays a hand, he looks at his cards, at his opponents, considers the betting, and makes an educated guess whether to check, bet or call, raise or fold [4]. And even if you know that you are going to bet or raise you also have to determine the amount you want to bet or raise.

Another important part of No Limit Hold 'em is, like in each form of poker, bluffing. In NL hold 'em you don't have to have the best hand to win the pot. When you make everyone fold their hand you win the pot without showing your hand. The type of opponents that you are playing is a huge part of deciding when and if you need to bluff. Bluffing will be discussed in the next section. But this also means that people will try to bluff you, and how can you tell if someone is bluffing you or actually has a good hand? In real life games you can sometimes have tells (physical motions that give away their hand) on someone, if they for example always look away when they bluff. In online poker you don't have any of these tells. The only thing you know about your opponents is their betting pattern, but when you have enough data on this you can classify your opponents and give an educated guess on whether they are bluffing or actualy have a hand.

In poker there are different poker combinations (later discussed in Hand evaluations). These combinations always consist of five cards. This sometimes gives some confusion. When for example the community cards are $T\diamondsuit 9\heartsuit 8\clubsuit 6\clubsuit 2\diamondsuit$ and each player holds a 7 in their hands their other card is unused. This will give a split pot and each player gets half of the pot. Here are the different poker combinations with an example (from high to low):

Royal Flush	$A\spadesuit K\spadesuit Q\spadesuit J\spadesuit T\spadesuit$
Straight Flush	$8\spadesuit 7\spadesuit 6\spadesuit 5\spadesuit 4\spadesuit$
Four of a Kind	$9\spadesuit 9\clubsuit 9\diamondsuit 9\heartsuit 5\clubsuit$
Full House	$A\clubsuit A\heartsuit A\spadesuit 2\heartsuit 2\diamondsuit$
Flush	$K\heartsuit 9\heartsuit 8\heartsuit 4\heartsuit 2\heartsuit$
Straight	$Q\clubsuit J\heartsuit T\heartsuit 9\diamondsuit 8\spadesuit$
Three of a Kind	$5\clubsuit 5\heartsuit 5\spadesuit K\heartsuit J\diamondsuit$
Two pair	$K\clubsuit K\heartsuit 7\spadesuit 7\heartsuit 3\diamondsuit$
One pair	$A\clubsuit A\heartsuit 8\spadesuit 6\heartsuit 2\diamondsuit$
High Card	$A\clubsuit Q\heartsuit 9\spadesuit 6\heartsuit 4\diamondsuit$

With NL hold 'em there are cash-games and tournaments. There is a big difference in strategy. The best tournament players are usually not very successful in cash-games and vice versa. With cash-games you have to buy-in for a certain amount. There is usually a minimum and maximum buy-in at cash-games and then you can play with that money. In a tournament there is a fixed buy-in amount where the top percent of the players is paid. Usually the top 10 % is paid. The tournament is where the big money lies because prize money for the first place is usually almost a quarter of the total prize pool.

We will be concentrating on tournament play and especially Sit and Go's. A normal tournament is more than one table and the top 10 % is paid. In a Sit and Go there is a maximum of one table, it can either be 2, 6, 8 or 10 players. The advantage of using this as a starting point is that you don't have to consider merging tables and so on. Another reason we took this approach is because it is relatively easy to expand to multi-table tournaments. Eventually it could also be extended for playing cash games.

3 The pokerbot

The idea of this part of the project is to create a pokerbot who can win against basic opponents. The different type of opponents will be described in the experiment section. This section contains the ideas we used to create our pokerbot. This section has mostly been done by Robin van der Zwan. In building this pokerbot we want to look at the following points to determine the play:

1. What is the status of the tournament?
2. How many players are at your table?
3. Who are the players at your table?
4. How does your stack compare to the blinds and ante's?
5. How big are the other stacks at your table?
6. Were do you sit in relation to the aggressive and passive players?
7. What bets have been made before you have to act?
8. How many active players are left after you act?
9. How much are the pot odds?
10. What is your position at the table (after the flop)?
11. What are your cards?

This seems like a lot of points and it is, but NL hold 'em is not just a simple card game. It requires quite an amount of effort to just play decent. If you are wondering why bluffing is not a part of the list, it actually is but it is part of the status of the tournament and part of who the other players are.

All of these points are very important and that is why we have to implement as many of them as possible. You miss a lot of information when you don't use everything you have got. In the following subsections all of the above points are discussed and we describe how we implemented them in our program. Each point is important but some points are more important than others. Because of the work involved with implementing each of these points, some points have not been implemented. This does not mean they are not important.

3.1 The status of the tournament

Most tournaments pay about 5–10 percent of the players. The Sit and Go tournaments that we concentrate on pay the first three places. This means that when there are ten players the first seven people out don't win anything. The prize pool distribution is as follows:

- 1st Place is 50 %.

- 2nd Place is 30 %.
- 3rd Place is 20 %.

So there is a big difference between getting 1st place and the other paying places. Usually play is normal until the paying places start to get close. Most players play conservatively to preserve their chips to be guaranteed at least some money. Good players become aggressive and change their play to get some easy chips, to make sure they don't just win some money but try to win the most, because people will try to avoid confrontations because that endangers them from getting in the money.

This means that our program should change its play if there are 4 or 5 people left. We are all ready changing our play at this point because the blinds are much higher but this is one of those parts we didn't have enough time to implement, mostly because these are very specific cases where for example if you have a big stack and there are 2 short stacks your actions change a little. However you also have to look at the history of the last few hands to get an optimized use of the changed play. Also unchanged play will still be decent/profitable play, just not the most profitable.

3.2 The number of players at your table

This is very simple to determine of course, but it is very important. If there are fewer people on the table you can/have to play more starting hands because there are fewer players in the hand. Which means that there are fewer good hands 'out there'. Also instead of paying for the blinds once every nine or ten hands now you have to pay them every three or four hands. This means you have to play more hands to just stay alive. Our program checks how many players are left in the tournament and makes different decisions if there are less people at the table. We experimented with this (when to change your play), see the experiment section.

3.3 How is your stack compared to the blinds

You don't look at the absolute size of the blinds and ante's. Their size compared to your stack is what is important. If the blinds are small compared to your stack you can survive many hands without playing. However, if your stack is just a few times larger than the blinds you have to start playing more hands. You can't wait for the premium hands. Hands you wouldn't even consider playing at the beginning of a tournament have to be played to survive.

In the Sit and Go tournaments, when this stage of the game is reached, usually half of the people are already out and you would have to change your strategy anyway. In the multi table tournaments this is not always the case. You might still be on a table with ten people when your stack is only a couple of times the blinds.

You can describe the value of your stack in multiples of the blinds. If you divide your stack by the big blind + the small blind you get a number we call M.

If your M drops under 5 you are in serious trouble and have to change your play dramatically. We have also implemented this. For example if $M = 5$, suppose you get dealt 9–9. When this happens at the beginning of the tournament (when your M is still high) you would just raise 2 or 3 times the big blind. Instead you would now have to go all-in. Our program calculates his M before each hand and takes different actions depending on its value.

There are also advantages of having a big stack because you can bully around the other players. They are in danger if losing their entire stack when they want to mess with you, while you can just lose chips but still remain in the tournament.

3.4 What are the other stack sizes at your table

In the previous subsection it became clear how important your stack size is. If your stack is larger than the other stacks on the table you can dominate the action. If your stack is average you can bully the small stacks but have to be careful for the big stacks because they can eliminate you. When you have a short stack (one of the smallest stacks on the table) you can't steal very often and just have to hope you win some showdowns with all your chips in the middle so you can acquire a larger stack.

This also means that if a player who is the short stack moves all-in he doesn't have to have as good a hand as a player who has a larger stack. You can now call with more hands than you usually would have called with. Our program will look at the M of the player who moved all-in and if his M is lower then 3 we will call where we would usually fold. There can be done a whole lot more with adjusting play according to the stack sizes but that will require huge amounts of code and the advantage you gain is not as important as some other aspects we did implement.

3.5 The type of players at your table

Originally this was planned to be a big part of our program. The idea was to use neural networks to classify each opponents aggressiveness and tightness to get a better idea of what certain raises mean. It is a whole different story if you are up against a player who plays every hand (loose player) than if you're up against a player who you know only plays good cards (tight player). If you are at a table with a lot of aggressive players you want to play conservative and if a table has a lot of conservative players you want to play aggressive. We were unable to implement this because the opponent modelling turned out to be very difficult. The problem is getting correct data to use for opponent modelling. The best opponent modelling would remember everything the opponents have done. It is a shame we couldn't implement this because it is an important part of playing No Limit Texas Hold'em. Our idea was to remember specific moves and determine the aggressiveness and tightness of the players. A specific move is, for example, trying to steal the blinds from the button position.

3.6 Where are the other players

When you are playing in a tournament you want the aggressive players to be on your right and the tight players to be on your left. This is not something you can just decide, but you can adjust your play if you have aggressive players to your right. If you have an aggressive player on your left you want to play less hands but if he is to your right you like to play more hands because you get to act after he does.

In the previous subsection it is explained why opponent modelling was not implemented. That means that the ideas in this subsection is also not implemented because to change play according to the location of aggressive and tight players you first have to know who is tight or aggressive.

3.7 What is the action before you act

The only absolute strong hands pre-flop is a pair of Aces. All other hands have to be evaluated in terms of the betting that has already occurred. A pair of jacks is good when there is no-one before you who has entered the pot. But when there has already been a bet–raise–re-raise it is likely to be second or even third best.

We implemented this to look whether the pot has been opened yet. When it's unopened raise with certain hands you would call or even fold with if the pot had been opened.

3.8 How many players are active after you act

There are different situations where you can either close the betting round or sometimes can't end the betting round. If for example there has been a bet and a raise in front of you, you can't stop the action regardless of your action because the player who made the first bet can still decide to raise. You have to play more careful if you're not sure of the action behind you. This is just something you have to take in consideration and always remains the same, no matter the opponent or the stage of the tournament. This is fairly standard to program simply because it is always the case.

3.9 How much are the pot odds

This is straightforward mathematics. What are the odds of making the hand (when you are drawing to catch a straight or a flush for example) and what is the amount you have to pay to stay in the hand? We implemented this as follows: First we look how many cards there are to make our hand. If we have 4 card to a flush (just need one more card to hit our flush) there are 9 cards that will give us that flush. When this is on the flop there are $52 - 5 = 47$ unknown cards. This gives a chance of $9/47$ that we hit our flush. If we have to pay less then $9/47$ of the pot we will call and otherwise we will fold. This can be changed because if we hit the flush you might get more money out of

your opponent (called the implied odds). See the mathematics section for more detail on how exactly those odds are calculated.

3.10 What is your position in relation to the dealer

It is an disadvantage if you have to act first in a hand, because you have to act with no information about what your opponent might do. It is good to act last because you get to see what your opponent(s) did. Some players even make moves to make sure they have good position. You might play weaker hands if you know that you have position. We didn't bother too much about this. It is more for improving our program later on.

3.11 What are your cards

Last but not least, what kind of cards are you holding? They of course are important. But all the aspects of a hand (from the previous subsections) are important. Sometimes you will be in a situation where you will make a play regardless of the cards that you are holding.

At the beginning of our program the hole cards that you get dealt are classified. We have 27 different classifications where 1 is the best classification and 27 the worst. Some card combinations get the same classification. If you get a pair of threes or a pair of fours you would like to play them in the same way so they will get the same classification. All cards you don't want to play will get classification 27. This does not mean you will never play those cards. There are situations where you will play any hand even if it has classification 27. These classifications are used to determine whether you will fold/call/raise. We can also experiment with this and we have done that.

3.12 Mathematics involved

Creating a computer program which can play (decent) poker requires a lot of calculations to determine what action has to be taken. In this section most of the mathematics that is involved is discussed.

3.12.1 Starting hands

We begin with the starting hands. Each player is dealt a starting hand that consists of 2 cards. Because a normal deck of cards is used there are $\binom{52}{2} = 1,326$ different starting hands. However the starting hand A♣K♣ can be considered to be the same as A♢K♢, just like 5♠5♡ can be considered the same as 5♢5♣. After removing all equal starting hands we are left with "only" 169 starting hands. We get this number by taking all pairs (13) and each combination of cards from the same suit (78 = 12 (A-x)+11 (K-x)+10 (Q-x)+9 (J-x)+8 (T-x)+7 (9-x)+6 (8-x)+5 (7-x)+4 (6-x)+3 (5-x)+2 (4-x)+1 (3-x)) and unsuited (78). This gives the idea you can say that you have a 1/169 chance to get Aces as a starting hand, which is incorrect. There are 6 ways to get dealt AA (A♣A♢, A♣A♡,

A♣A♠, A♢A♡, A♢A♠, A♡A♠) which gives $6/1326 \neq 1/169$. However we can still use just the 169 starting hands instead of the 1,326 if we don't care about the chance of getting a particular starting hand. It can be useful if we try to determine what an opponent might be holding.

The next part of the game where new cards come into play is the flop. We already know our own two cards and they can't be part of the flop. This means there are $\binom{50}{3} = 19,600$ different flops possible. For the turn there are $\binom{47}{1} = 47$ possibilities and finally for the river there are $\binom{46}{1} = 46$ possibilities. This gives a total of

$$1,326 \times 19,600 \times 47 \times 46 \approx 5.6 \times 10^{10}$$

possible outcomes of each game if we only look at our own hand and don't take any opponents into consideration. We could deal the cards 50 billion times and have not once see the exact same 7 cards.

First we wanted to calculate all the odds for each possible starting hand to win before the flop, after the flop, after the turn. And this has to be done against one, two, three, four, ... opponents. We decided not to brute force these calculations for several reasons. First of all it would take to long to calculate even though it would only have to be calculated once. With just one opponent you would have to calculate the following amount of different possibilities:

$$\binom{52}{2} \times \binom{50}{2} \times \binom{48}{3} \times \binom{45}{1} \times \binom{44}{1} \approx 5.56 \times 10^{13}$$

And with more opponents the number gets even larger. This is however not the only reason we decided not to calculate each of the odds precisely. It doesn't make much of a difference whether you know the odds are $58,392\%$ or $58,4\%$. Those small percentages don't contribute to the calculations of making the decision whether to fold/call/raise. Position is for example much more important than to know the exact odds.

However, after some searching we found an approximation of the probability of winning for any given hand and specific flop, against varying numbers of opponents (1,3,6,9). They were computed by considering each of the 169 possible hold card combinations, and the $50 \times 49 \times 48/6$ (you divide by six because it doesn't matter whether the flop is AKQ or QKA) possible flops. In each of the resulting 3,312,400 cases, we imagine dealing the turn and river cards randomly and then computing the probability of beating N opponents who all hold independently chosen random hands. This means that for each of the 3 million cases, for the case of one opponent, we compute:

$$(1/1081) \times \sum_{T=1}^{1081} (1/990) \sum_{H=1}^{990} I(T, H)$$

where T indexes the $47 \times 46/2 = 1081$ possibilities for the turn and river cards, H indexes the $45 \times 44/2 = 990$ possible hole combinations the opponent can

have, and $I(T, H)$ is 1 if we have the better hand given these choices for the turn, river and opposing hand, 0 if we lose and 0.5 if we tie. So for each hand you look at the possible turn and river cards, for each turn and river you look at each possible opponent's hand, for each opponent's hand we see who wins. To handle more than one opponent, we make the approximation that opponent's hands are independent after the board cards are dealt (which isn't perfectly true — if the board has 3 aces, then they can't both hold the fourth), when the computation for N opponents becomes:

$$(1/1081) \times \sum_{T=1}^{1081} [(1/990) \sum_{H=1}^{990} I(T, H)]^N.$$

However, we will use this at most against three opponents and just start of using it against one opponent. So N is at most 3. This gives a generated file of 3,312,400 lines for each starting hand with each possible flop, and the probabilities to win with it.

3.12.2 Pot odds

This section is about calculations that actually take place during the game. Because not all the cards are shown at the beginning of a hand sometimes you have to calculate what chance you have to make a hand. The following example will show some calculations you have to make during the game. This is the only mathematics involved during the gameplay but it is important when you have to decide if you are going to play a hand.

You are in late position. The blinds are \$5–\$10. A person who is in early position raised three times the big blind. There are two callers from weak players. When it is your turn to act you have the hand 7♣8♣. This is a decent hand to play from this position with already 3 people in the hand. So you call, the small blind folds and the big blind calls. The pot is now \$30 + \$30 + \$30 + \$30 + \$5 + \$30 = \$155.

The flop is 6♣T♢K♣. The Big blind bets \$75 (about half the pot), the player in EP calls and the other two players fold. Now it is your turn to act. You have to put in \$75 to win \$155 + \$75 + \$75 = \$305. This is a little more then 4–1 odds. We assume that if either a 9 (to give you a straight) or a club (to give you a flush) will give you the winning hand. This means there are 4 + 8 = 12 cards in the remaining 47 cards that will give you the winning hand. This means there is a 12/47 = 25.5% chance to win. That is a little more then 1/4. Because the odds of making your hand is higher than the pot odds, we will put in \$75. The pot now holds \$380.

The turn card is 5♡. The big blind bets \$100 and the person in early position calls. We have to pay \$100 to win \$580. And our odds of making our hand is even better. Now any 4 or any 9 or any club will give us the winning hand. Those are 4 + 4 + 7 = 15 cards that help us out of the 46 cards left. That is almost 1/3 and we only have almost 6–1 pot odds. Which makes this a call.

Sometimes even when you don't have the correct odds to call you can still make the call because you expect to make money in the next betting round if you made your hand. Those are called the implied odds. We don't take those in consideration in our program. There are also situations where you would bet a draw to either win it at that moment or if you don't you can still win if you hit your draw. This is called a semi-bluff and is very powerful in No Limit Texas Hold'em.

The following table shows the number of outs for the most occurring situations and important:

Drawing Hand	Drawing to	Number of Outs
Pair	Three of a Kind	2 outs
Two pair, inside straight	Full house, Straight	4 outs
Two overcards	Pair	6 outs
Open-ended straight draw	Straight	8 outs
4-card Flush draw	Flush	9 outs
Flush draw with inside straight draw	Flush or Straight	12 outs
Flush draw with open-ended straight draw	Flush or Straight	15 outs

As shown in the example above, when you have the number of outs you can easily calculate if you should fold or call (Not taking implied odds into consideration.).

3.12.3 Hand evaluator

There are different stages where we need to evaluate what kind of hand we currently hold using our own cards and the community cards. At the flop we look at the available five cards to determine if we have a pair, a flush, a straight etc. This happens again at the turn but then we use the available six cards. Finally at the river we use all seven cards to determine our hand and finally if there is a showdown to determine who is the winner we evaluate all hands that are involved in the showdown to determine the winner.

We use a very simple method to evaluate the hand we have. We simply look at the available cards and use basic comparisons to find pairs and straights. We also check for the flush but do that separately. Because this is such a big project we didn't put much time in this part.

There has been done some related work on hand evaluations by Kevin Suffecool [15]. He first distinguished the poker hands to go from $\binom{52}{5} = 2{,}598{,}960$ unique hands to 7,462 distinct hands. He realized that even though there are nearly 2.6 million unique hands, many of those hands actually have the same poker hand value. In other words, somebody holding an AJ942 flush in spades has the exact same value hand as somebody with an AJ942 flush in clubs. His results are shown in the following table. Where 'unique' means the actual number of hands and 'distinct' means the number of different kinds of hands. For example, consider the straight flushes 3♣4♣5♣6♣7♣ and 3♡4♡5♡6♡7♡. They are 2 unique hands, different cards are used in each hand, but they can be con-

sidered to be the same hand because they have an equal value.

Hand Value	Unique	Distinct
Straight Flush	40	10
Four of a Kind	624	156
Full House	3744	156
Flush	5108	1277
Straight	10200	10
Three of a Kind	54912	858
Two pair	123552	858
One pair	1098240	2860
High card	1302540	1277
TOTAL	2598960	7462

To complicate matters, the algorithm needed to be order independent. For example, if the cards $K\diamondsuit Q\spadesuit J\clubsuit T\heartsuit 9\spadesuit$ are passed to his evaluator, it must generate the value 1601 (which is the value that is assigned to a King High Straight). However, if the order of the cards is changed in any fashion, it must still return the value of 1601. Mixing up the five cards does not change the overall value of the hand. At first, he could always simply sort the hand first before passing it to the evaluator; but sorting takes time, and the last thing you want to do is waste CPU cycles. So instead he would use prime numbers. He assigned a prime number to each card.

Two	2
Three	3
Four	5
Five	7
Six	11
Seven	13
Eight	17
Nine	19
Ten	23
Jack	29
Queen	31
King	37
Ace	41

If you multiply the prime values of the rank of each card in your hand, you get a unique product, regardless of the order of the five cards. A King High Straight hand will always generate a product value of 14,535,931. Since multiplication is one of the fastest calculations a computer can make, this method is much faster than sorting.

This way he got one of the fastest hand evaluators. And he made his code open source so we can use it. Implementing this completely is something for future work on this project.

3.13 How the bot works

In this section we give a general description on how the bot works, in the following section we present a more detailed description on what the bot makes its decisions.

A tough part about this project is how and where to begin because it is such a huge project. The first thing was to give a classification to the starting hands. Each hand is categorized so that the correct action can be taken. The best hands are of course aces and weak cards like 2 and 7 of different suits are given the lowest classification. The first part where the bot has to take action is pre-flop when only the hole cards are known. We use the classification-value, the position and the actions of the other players to decide what to do.

The input for the bot first is the position of the button to decide what the position of the bot is. It also gets the stack sizes of all the opponents. The cards are dealt and the bot gets his two hole cards. It will give a value to that hand according to how good a hand it is. The next input for the bot consists of the actions of each player. When it is time to decide whether or not to play the hand, he will use the input up to that point. The output will be an action like fold, call or raise. When raising, the bot also gives an amount as part of its output. After its action it gets input on what the remaining opponents actions are. This repeats after the flop is dealt and again after the turn and river card, where of course the input consists of the community card(s) instead of the hole cards. The final input is about the player who wins the pot. Here an output can be either to show or muck your cards.

The program can be viewed as a poker playing agent or bot. During a complete hand there are several moments where the agent has to take action. His action can either be fold, check, call or raise, where a raise also comes with an amount. An action is based on all information that is currently available to the agent. As mentioned before this only concerns the current hand. First of all the agents uses a rule-based algorithm (cf. Section 3.11 and Section 3.12.3) to determine the strength of his hand (i.e., the community cards and the hole cards together). Then the actions of the other players so far are taken into account. Again, a rule-based system (cf. Section 3.7) decides what to do, using both the output from this and the first step. Finally, for the second step we use some randomness to vary the play of the bot.

When playing good opponents it is important that you don't always do the same thing with the same cards. If you always raise a certain amount from the first position when you have AA it will soon give away your hand. For this reason we have included some randomization in our actions. For example with AA in early position we use a random generator where in 25% of the time we call and 75% of the time we raise. And we also randomize our raises. Against the opponents that we are playing for the experiments this is not crucial but

when we want to expand our project it is important to have built in already. We have a function called `outs` to calculate our odds to call or even raise (also see the previous section on how we used this).

In poker you have multiple competing agents that must deal with probabilistic knowledge, risk management, deception, and opponent modelling, among other things. This seems a lot to take care of and it is. But that is what a poker playing agent requires to become decent enough to compete with real players.

When we started with the project our idea was to create an agent who could adapt itself to the different type of opponents. The agent would have a predefined set of rules on how to act in different situations but it would be able to detect what the current situation was. This so-called reflex based agent [14] observes what the actions around him are and decides, using his own set of rules, the appropriate actions. This agent structure seems to be the most natural choice because the rules and possibilities in poker are known and don't change. The only thing that changes is the way people play certain hands. For that you'll need good opponent modelling. In a future project more advanced strategies can be implemented for the bot instead of the simple reflex based agent that we now have. Our first intentions were to use some sort of neural network for the opponent modelling but this seemed unfeasible. However some variation can be added to the project in the future.

4 Pokerbot Experiments

In order to see whether our pokerbot will actually win against basic opponents we have to test it of course. We also created a couple of basic bots to test against. In the next section we give the results against those bots. We tested our program against the following bots. These bots are created under the assumption that on line beginning players often play very bad, so that you will indeed encounter these kind of opponents.

- Call_bot, a bot who never bets but always calls your bets (unbluffable). Also called a calling station. Often plays in low no-limit poker on line.
- Fold_bot, a bot who folds every hand.
- Raise_bot, a bot who raises every hand. Also often found in low no-limit poker on line.
- Random_bot, a bot who does random actions. Often beginners play like Random_bot would. However the bot won't fold when it has the option of checking.
- Simple_bot, a bot who can make some simple decisions from the previous sections to determine his play.

Simple_bot is the bot that we created that can use some of the basic rules to play winning poker. First when its cards are received it gives a classification

to his hand. He will then see what the action is before he is to act. Depending on the action and the hand classification it is determined what kind of action has to be taken. Simple_bot uses a randomized function to have little variations to its play. When determining its move pre-flop it will also take the number of opponents that are left into consideration. It does not take chip stacks of the opponents into consideration and also does not take the type of opponents into deciding its action.

After the flop is dealt it will see how strong his hand is and does the action that is expected with its hand strength. Again with a randomized function to have little variations from time to time. The possible hands that are possible on the flop are not taken into consideration when deciding its action. This is still one of the flaws but requires quite a lot of work to implement.

The action after the turn and river are the same as the action after the flop. It is also determined by the strength of the hand. The bot could be improved a lot if it would take the possible hands into consideration. This could be done in the future but will be a lot of work. There are also a lot of different features that still have to be implemented, for example taking chip stacks into consideration, taking the table image into consideration and much more.

Against each type of opponent we've let our bot play 100,000 games, but also against combinations of opponents. Since a table consists of ten players we have also experimented with 4 Random_bots, 2 Raise_bots and 3 Call_bots for example.

The problem is of course that these are still very basic opponents and it is not an indication that our bot is a good player if it beats them, just that we can defeat the basic players.

5 Results

For the first experiment we used a full table, consisting of two bots of each type (Call, Fold, Raise, Random and Simple). Each bot starts with 200 in chips. We start with a big blind of 2. (In most real-money on line Sit and Go tournaments each player starts with 100 times the big blind as a stack.) We ran our program 100,000 times and the results are shown in the following table. The value of the cells is the number of times that bot finished in that position. For example, Simple_Bot1 finished 812 times as winner, 4791 times as second and so on.

	1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th
Simple_Bot 1	812	4791	38587	27274	5000	13886	9421	224	5	0
Simple_Bot 2	804	4637	38667	27186	5090	14066	9282	229	6	0
Fold_Bot 1	0	104	2478	17468	44430	30028	5492	0	0	0
Fold_Bot 2	0	107	2398	20337	42333	31958	2867	0	0	0
Call_Bot 1	24598	23211	4552	1890	700	1647	10043	9343	7684	16325
Call_Bot 2	25132	23041	4325	1916	726	1546	9789	9693	8945	14981
Raise_Bot 1	24636	22442	4546	1938	745	1863	9881	10311	10015	13527
Raise_Bot 2	24006	21655	4412	1960	702	2098	10311	10775	11141	12925
Random_bot 1	4	4	9	11	124	1370	16430	29328	30374	22345
Random_bot 2	8	10	26	20	150	1543	16484	30027	31830	19897

There are some interesting results from the first experiment. First of all we were disappointed in the performance of Simple_Bot, we expected for it to end higher on average. From the 100,000 games that were played it never finished as 10th. When we were trying to find the reason why it performed worse than expected we found the 'problem' was at the end of the tournament. Our bot folded to many hands, waiting for a good one, and thereby lost most of its chips because of the higher blinds at the end of each tournament. When we look at the results from Call_Bot and Raise_Bot it seems like they behave in a somewhat similar way. This was to be expected because they both play every hand to the end and it just comes to luck who finishes on a higher position. Fold_Bot performed better then expected by getting a 5th place on average. The worse bot was Random_Bot, we never expected it to finish high because random actions don't get you very far during a pokergame.

We modified the Simple_Bot so that it would perform better at heads-up play and overall better performance at the finishing stage of the tournament. This was done by lowering its hand standards and playing more aggressive, meaning it would raise its good hands higher than before. We redid the experiment with the same conditions but with our modified Simple_Bot and the following table shows the results of 100,000 runs.

	1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th
Simple_Bot 1	34271	19910	17231	8045	1905	9474	6540	2202	398	24
Simple_Bot 2	25586	20429	18824	10034	2320	12183	7834	2522	381	30
Fold_Bot 1	0	412	6724	31675	44285	16643	261	0	0	0
Fold_Bot 2	0	424	6082	33925	42570	16937	62	0	0	0
Call_Bot 1	12884	15943	11545	3391	926	6139	8753	7059	5168	28176
Call_Bot 2	12729	15755	11438	3268	904	6110	9017	7422	11419	21877
Raise_Bot 1	7311	13562	14375	5072	1607	7676	9759	9178	15064	16372
Raise_Bot 2	7187	13464	13669	4260	1964	8607	9911	11002	17074	12841
Random_bot 1	22	61	66	135	1701	7879	23463	29397	25781	11492
Random_bot 2	10	56	46	195	1818	8352	24400	31218	24715	9811

We can see from the results that the modification to Simple_Bot makes a

big difference. The performance of the bots, except for Simple_Bot, remains the same which was to be expected because we only modified Simple_Bot. Now when a few people are out of the tournament it changes its playing style and plays more hands and more aggressively. This is the best result so far and what we hoped for. Our bot can now defeat the simple type of opponents. This will probably be fine for the very low limits of on line poker, however when you want to play on higher limits it becomes a too predictable opponent because it is incapable to make moves which are not included in its program yet.

But what will happen when we start with a smaller or larger starting stack? For the following experiments we first set the starting stack to 500 and then to 50 to see whether it makes a difference. The idea is that when the starting stack is smaller there is more luck involved and when it is bigger skill becomes more important. The first table are the results from 100,000 tournaments with a starting stack of 500. The second table are the results from 100,000 tournaments with a starting stack of 50.

	1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th
Simple_Bot 1	38858	40387	10458	7829	82	816	1055	401	117	17
Simple_Bot 2	43721	23650	14640	13114	198	1935	1979	563	184	29
Fold_Bot 1	0	0	141	8054	44810	42752	4243	0	0	0
Fold_Bot 2	0	0	249	8341	45726	40853	4831	0	0	0
Call_Bot 1	7788	12243	15892	12639	2427	2560	16942	10997	8250	10260
Call_Bot 2	7930	12264	15731	12822	2412	2538	17184	10990	8007	210092
Raise_Bot 1	860	5833	21612	19397	2413	3321	16529	11321	8293	10431
Raise_Bot 2	843	5623	21277	17804	1887	4163	17297	11880	8675	10550
Random_bot 1	0	0	0	0	19	497	10002	26842	33204	29436
Random_bot 2	0	0	0	0	26	565	9938	27006	33280	29185

	1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th
Simple_Bot 1	19689	20633	8200	12456	23896	7792	1983	516	146	4689
Simple_Bot 2	18439	22034	6866	15219	23390	7092	1861	487	417	4469
Fold_Bot 1	0	14973	41586	33309	9710	422	0	0	0	0
Fold_Bot 2	0	17034	39959	33927	8561	519	0	0	0	0
Call_Bot 1	14517	5139	720	173	599	925	982	1140	7559	68246
Call_Bot 2	14688	5163	626	177	666	1052	1403	6180	53029	17016
Raise_Bot 1	13152	5609	748	403	1168	1766	4639	40424	29755	2346
Raise_Bot 2	13128	5344	597	504	1412	3508	30179	39141	5216	971
Random_bot 1	3206	2170	343	1519	9871	32982	38662	7633	1934	1680
Random_bot 2	3181	2085	355	2313	20727	43942	20291	4479	1944	583

We can see a big difference in results from both experiments. When we start with a much larger chip stack Simple_Bot wins more then 80% of the time but when the chip stacks are very small it will win less then 40% of the time. This shows that the longer the tournament takes the more advantage Simple_Bot gets because it has enough time to wait for decent hands. The fewer amount of starting chips the more it becomes like a game of luck instead of skill.

In the last experiment we only used one Simple_Bot and an extra Raise_Bot. The starting stack is again 200. In the second table in this section we've seen that with two Simple_Bot's in 60% of the cases a Simple_Bot wins. It would also be interesting to see if there is only one Simple_Bot if it can also win 60% of the time. The following table shows the results from 100,000 runs with one Simple_Bot replaced by a Call_Bot.

	1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th
Simple_Bot 1	48949	17800	13931	887	2641	8064	5894	1644	210	10
Fold_Bot 1	0	420	9904	45232	39702	4736	6	0	0	0
Fold_Bot 2	0	440	9762	45922	40593	3271	12	0	0	0
Call_Bot 1	11335	15452	12852	1567	1219	8461	8623	7911	5242	27347
Call_Bot 2	10968	16188	13063	1629	1206	8392	8398	7708	11181	21273
Call_Bot 3	11107	16390	13140	1575	1181	8718	8718	9061	14642	15456
Raise_Bot 1	8785	16703	13612	1506	1752	8879	9477	11419	15913	11950
Raise_Bot 2	8760	16196	12859	1114	2095	8926	9786	13092	15719	11446
Random_bot 1	45	193	479	256	4574	19996	24115	23805	19477	7060
Random_bot 2	51	218	298	312	5037	20579	24971	25360	17616	5458

The performance of the basic bots remains the same and Simple_Bot would win almost 50% of the time.

Because Fold_bot did much better than expected in the experiments we decided to test how well it would work in real-money games. We wanted to research this because we figured that real-life players act different than the basic bots we created. The tournaments that were used for this experiments were $6 buy-in tournaments with 10 people and where the top 3 get paid. There was a standard payout structure which means that first place gets 50%, second place gets 30% and third place gets 20%. We tried the normal tournaments where the blinds go up every 10 minutes and speed tournaments where they go up every 5 minutes. The results are in the following table.

Table Number	Type of tournament	Finishing Place
125825	Normal	4th
126502	Normal	4th
1304468	Speed	5th
1304530	Speed	4th
126511	Normal	5th
1304686	Speed	5th
1304322	Speed	4th
125943	Normal	3th
126554	Normal	4th
1304654	Speed	5th

It is surprising to see that you can always survive more than half of the people by just folding every hand. And even on one occasion folding got into the money. We expected human players to notice we would fold every hand. According to

the results this wasn't often the case. This strengthens our assumption that on line poker rooms are full of bad players which can be easily defeated by a pokerbot who follows even simple rules.

6 Data mining

In this section we will apply data mining to a database of poker games. From [17] we quote:

> Data mining is the extraction of implicit, previously unknown, and potentially useful information from data.

In this case our data consists of a database of hands played by people on Internet Relay Chat (IRC) over a large period of time. In the past, before on line gambling arose, a lot of people used to play poker for free on the IRC poker server. A program called Observer was written by Michael Maurer. This program sat in on poker games on this server and logged every game it witnessed. From [9] we quote:

> This resulted in the collection of the more than 10 million complete hands of poker (from 1995–2001) that constitute the IRC Poker Database.

This program observed the game from a real observer's point of view, which means a lot of data is never recorded. For example, most of the cards dealt to players are never revealed and as such are not included in the database. This poses some problems which will be explained later on. The upside is that the database does include a lot of information, be it explicit or implicit, and contains data recorded over almost seven years.

The goal of this data mining project is to discover new information and gain new insights in the way players play. We hope to find certain patterns in player actions which can in turn be used to better "read" players. For example, we may find that players who raise by less than a certain percentage of their stack will fold the next round in most cases. This could be put to use by adding an exception to the bot's normal actions. Normally this raise might have lead to our bot folding, but if we tell our bot to ignore raises that are small enough to fit in the pattern we found our bot might instead keep playing and go on to win the hand. There are no guarantees of course. Sometimes a cautious player with a strong hand may make such a small raise and go on to win the hand. And even if the player who made the raise folds another player may win. However, in the long run the added exception may make a difference. Implementing and testing these exceptions is a lot of work and beyond the scope of this project.

6.1 The database

As mentioned above we will use the IRC Poker Database created using Michael Maurer's Observer program. This database consists of a large number of files.

For every month there is a tarball of files containing all information recorded during that month. This tarball contains one file named HDB, which contains data directly related to the hands played. In this file there is a line for every hand which provides information on the number of player, the cards on the table and such. There is another file named HROSTER which states the names of all players involved in every hand. Finally there is a file for every player who played poker during the month in question. These files contain information like the winnings, the cards and the player's actions for every hand and are named PDB.playername.

The following tables from the IRC Poker Database website [9] specify the file formats.

HDB file format	
column 1	timestamp (supposed to be a unique integer)
column 2	game set # (incremented when column 3 resets)
column 3	game # reported by dealer bot
column 4	number of players dealt cards
column 5	number of players who see the flop
column 6	pot size at beginning of flop
column 7	number of players who see the flop
column 8	pot size at beginning of turn
column 9	number of players who see the flop
column 10	pot size at beginning of river
column 11	number of players who see the flop
column 12	pot size at showdown
column 13+	cards on board (0, 3, 4 or 5)

HROSTER file format	
column 1	timestamp
column 2	number of player dealt cards
column 3+	player nicknames

PDB file format	
column 1	player nickname
column 2	timestamp of this hand (see HDB)
column 3	number of player dealt cards
column 4	position of player (starting at 1, in order of cards received)
column 5	betting action pre-flop (see below)
column 6	betting action on flop (see below)
column 7	betting action on turn (see below)
column 8	betting action on river (see below)
column 9	player's bankroll at start of hand
column 10	total action of player during hand
column 11	amount of pot won by player
column 12+	pocket cards of player (if revealed at showdown)

Betting action encoding	
-	no action; player is no longer contesting pot
B	blind bet
f	fold
k	check
b	bet
c	call
r	raise
A	all-in
Q	quits game
K	kicked from game

What follows is an example of the lines from each file for a certain hand. The following line was found in an HDB file:

1	2	3	4	5	6	7	8	9
820483415	2885	5	6	2/75	2/135	2/235	2/435	6s 2c Js 2s Qc

The first number is the timestamp, this should be a unique value identifying the hand. The second and third number are the game set number and this particular game's number within the set. The game set increases when the third column resets. The fourth number indicates the number of players that are dealt cards, six in this case. The fifth and sixth number represent the number of players who see the flop and the stack size at the time of the flop respectively. Apparently four people folded before the flop, since only two players remained at the time of the flop. What follows are a similar pair of numbers for each following phase (turn, river, showdown). Finally the community cards are listed, in this case a six of spaces, a two of clubs, a jack of spades, a two of spades and a queen of clubs.

When we look up the same hand in HROSTER we find:

1	2	3	4	5	6	7	8
820483415	6	`BradR`	`Chimera`	`bmk`	`fatal`	`sagerbot`	`strider-1`

The timestamp is identical which means the data applies to one and the same hand. The second column lists the number of players. Once again we see there are six players involved. Then the players' names are listed. To find `BradR`'s actions we need to look for a line with timestamp 820483415 in the file PDB.BradR.

When we do so for each of the players we find the following lines in their respective files:

1	2	3	4	5	6	7	8	9	10	11	12
bmk	820483415	6	1	Bf	-	-	-	490	5	0	
sagerbot	820483415	6	2	Bf	-	-	-	415	10	0	
strider-1	820483415	6	3	cc	b	b	bc	435	210	0	Ts Th
BradR	820483415	6	4	f	-	-	-	650	0	0	
Chimera	820483415	6	5	f	-	-	-	525	0	0	
fatal	820483415	6	6	r	c	c	r	485	210	435	Ks Qs

These lines have already been ordered by the values in the fourth column, the player's position. Columns five through eight list each player's action per phase. Starting at column five we can now read how this hand was played.

bmk and sagerbot make their blinds.
strider-1 is the first to act, he calls.
BradR and Chimera both fold.
fatal raises, which leads to both bmk and sagerbot folding.
strider-1 calls again, which ends the pre-flop phase.
At this point the first three community cards hit the table, a six of spades, a two of clubs and a jack of spades (seen in HDB).
strider-1 is first to act after the flop, he bets and fatal calls.
The fourth community card, a two of spades, hits the table.
Once again strider-1 bets and fatal calls.
The final community card, a Queen of clubs, hits the table.
Once again strider-1 bets, but this time fatal raises and strider-1 calls.
strider-1 only has a pair of tens, meaning fatal wins with a flush.

Column nine shows each player's stack size before the hand, while column ten shows that player's "action" (chips placed in the pot) and column eleven shows each player's winnings. Columns 12 may contain the player's private cards, but this information is often not available. bmk started with 490 chips, he placed 5 in the pot (small blind) and won nothing, so he now has 485 chips left. fatal on the other hand gained chips. He started with 485 chips, he placed 210 in the pot and won 435 in the end, so he is left with 710 chips.

6.2 Data preparation

From [17] we quote:

> Preparing input for a data mining investigation usually consumes the bulk of the effort invested in he entire data mining process.

The best way to minimize the amount of work required to prepare the data is to make sure the data recorded is usable and stored in a useful format from the start. However, since we are working with an existing data set we don't have such a luxury. We will have to transform the data from its current form to a form we can use easily. The format of the existing data is described above.

One of the first things to consider is what information is contained in a single record. We want to find patterns in a player's actions, so we need to closely inspect every single action by itself. The existing database groups all actions for a hand together. A player's action only means something in a certain context, so we also need to reconstruct as much of the game state as possible.

We wrote a Python [2] script to perform this preparation. This script uses the MySQL for Python module [3] to insert the desired data into a relational database. To simplify later data mining we stored every action together with the game state at the time of the action in a single record.

The script has to read a line from both the HDB file and the HROSTER file and make sure they refer to the exact same hand by comparing the timestamp. The line from HROSTER contains the names of all players involved. The script then has to open the appropriate PDB files and find the line referring to the hand in question, once again by comparing time stamps. Once all these lines have been found the script simulates a game of poker. Instead of dealing cards, the cards given in the data files are assigned to players at the appropriate times and instead of requesting player actions, the actions are read from the data files. Right before any action is performed we store the current state of the game combined with the action to be performed in our MySQL database. The resulting script is roughly structured as follows:

```
for each line in HDB:
  read line x from HROSTER
  for each player in x:
    read line from PDB.playername
  initialize game state
  for each game phase:
    do:
      index = 0
      for each player:
        store action[player][index] and the game state in out databse
        update game state
      index++
    while more actions required
```

The updating of the game state involves keeping track of stack sizes, the number of players who have folded, the number of raises etc. For any action a player takes we want to store that action and as much relevant data as we can find. To return to our previous example, we start by initializing the game state. This means we set the pot to zero, we set each player's stack size to the value found in their file, we clear all cards and reset all variables used to track the actions per hand. In our example, the first player to act is `strider-1`. We create a new record which says `strider-1` called before the flop, after 0 people had called, 0 people had raised, 0 people had folded, that the pot size at the start of the phase was 0, 6 players were still active, he didn't have any knowledge of the community cards but he had a $T\spadesuit$ and $T\heartsuit$ in hand. We update the game state to reflect that there has been 1 call.

The next player to act is `BradR`. He folds, so the create a record which says that `BradR` folded before the flop, after 1 person had called, 0 people had folded, and all the other information stored in the previous record, except that in this case we don't know his private cards. We update the game state again to reflect that there has also been 1 fold.

The next player to act is `Chimera`. He also folds, so the next record we create is identical to the last one, except for the player name and the number of people who folded before, which is 1 in this case.

This step is repeated until there are no more actions in a phase. This same loop is then repeated for other phases. After the last phase we clear the game state and repeat the process for the next hand. An example of a record created this way can be found in the next section.

As mentioned before, we only have data available to outside observers, so in most cases we do not know which cards an individual player had. If a hand comes down to a showdown we obviously see the hands of all players involved. But if a player folds earlier than the showdown their hands are not revealed because this information could have serious consequences for the remaining players. If a hand doesn't come down to a showdown the winning player is the only one who didn't fold and he or she may choose to reveal his hand. Some people do this to show off their bluff or their luck, others choose not to let other players in on any unnecessary information. Missing this information is obviously a big disadvantage for us, as the cards a player is dealt are one of the biggest factors in whether or not they choose to play.

Another important piece of important information that is missing is the current size of the pot. For every phase of the hand (pre-flop, flop, turn and river) we know how large the pot was at the start of the phase. But after someone bets or raises we no longer know exactly how large the pot is when the next player gets to make a move. This information can be very important because the pot odds can make a somewhat weak hand interesting to play. For example, if a player estimates his chance of winning at about 20%, he has to put 200 in the pot to play and the pot contains 1500 the player may very well choose to play. He is most likely to loose the 200 chips, but the profit of winning (1500:200) outweighs the chance of losing (4:1).

Other data should always be available for every hand. However, databases often contain some unexplained errors or gaps. One example of data that might be missing in the IRC Poker Database is player data. In some rare cases the file containing a player's actions may not contain information for some number of hands. This can be due to some kind of error or deliberate removal, but the fact remains that the information is not there. There are two ways of dealing with this specific problem.

1. Ignore the player in question in all the hands for which his data is missing.

2. Ignore all hands in which the data for any of the players is missing.

With option 1 we keep a larger amount of hands in our database which means we have more data to mine. However, if the actions of even a single player in a

given hand are missing this skews the data on the other players. We chose to just ignore the entire hand instead. That way we do not store potentially misleading data and with the amount of data available we need not worry about ignoring a few incomplete hands.

At this point we have a database containing a separate record for every action along with as much relevant information about the current state of the game as we could gather. This includes information like the number of players, the number of players who folded, the community cards at the moment and hopefully the cards the player is holding.

We decided to use the Weka data mining suite [12] to do the actual data mining. This suite provides a number of interfaces and algorithms that can be used immediately. Weka works with the arff file format, which is a fairly simple comma-separated file format, but can also work with certain database management systems. We decided to use the arff file format. It is very easy to extract specific data from an existing database and store it in arff format. We may not always want to include all information or records, so we can generate separate arff files containing slightly different data. For example, we can generate an arff file which includes only actions by specific players, or actions which lead to a person running out of chips.

We will also use DF [7], a depth first implementation of the Apriori algorithm, written by Wim Pijls and Walter Kosters. The Apriori algorithm finds frequent itemsets in a database. This means it looks for combinations of attributes that occur frequently. This algorithm takes a large number of itemsets and determines which subsets occur at least C times, where C is called the minimum support. For example, if our input itemsets consist of number sets and one of the itemsets is {1,3,7} this counts once towards each of the following subsets: {1}, {3}, {7}, {1, 3}, {1, 7}, {3, 7} and {1, 3, 7}. If another of the itemsets is {1,7} and the minimum support is two, than each of the following subsets are considered frequent itemsets: {1}, {7} and {1, 7}. For more information on the Apriori algorithm and the DF implementation, we refer to the paper [13] on this implementation.

6.3 Information in database

In preparing the data we simply copied some data, while other data was derived by simulating the flow of the game. What follows is a list of data we stored in the database we used for actual data mining, explained with an example record created from the example used before.

Field name	Example data	Explanation
id	75	Unique record id.
player	`strider-1`	The name of the player. There is a good chance we will not do anything with this field, but it might be possible to find a way to identify a player based on his or her actions.
phase	river	The current phase of the hand. This is very important as players act differently during the pre-flop than they do right before the showdown. The phases are enumerated in their chronological order (pre-flop, flop, turn, river).
potsize	235	The pot size at the start of the current phase. We are really interested in the current pot size, but as this information is not available we hope that pot size at the start of the phase, combined with the number of raises, calls, etc can somewhat replace the information we really wanted.
activeplayers	2	The number of players that are still participating in the hand.
inactiveplayers	4	The number of players that folded.
allins	0	The number of players that have gone all-in.
raises	2	The number of times someone raised.
hand1val	T	The value (1–9, T, J, Q, K) of the first private card.
hand1suit	s	The suit (c, d, h, s) of the first private card.
hand2val	T	The value of the second private card.
hand2suit	h	The suit of the second private card.
card1val	6	The value of the first community card.
card1suit	s	The suit of the first community card.
card2val	2	The value of the second community card
card2suit	c	The suit of the second community card.
card3val	J	The value of the third community card
card3suit	s	The suit of the third community card.
card4val	2	The value of the fourth community card
card4suit	s	The suit of the fourth community card.
card5val	Q	The value of the fifth community card
card5suit	c	The suit of the fifth community card.
action	c	The action the player takes in the current situation as described by the other attributes. Note that there can be multiple records for any situation if that situation occurs on separate occasions.

7 Data mining experiments

Given the database containing the information as explained above, we will use the Weka data mining suite [12] to try to find some patterns. Given the complexity of the problem we do not expect to find a pattern that tells us how

people act and just trying to run different algorithms with different settings on the data could produce some results, but this doesn't seem very likely. Instead we will pose some questions for which we will try to find some answers using different algorithms.

We will try to find some basic, but potentially interesting statistics, like the percentage of folds in each phase of the game.

We will look at player's behaviour in relation to the number of active players. One would expect people to be more cautious and fold more often in games with more players, but a larger number of players also means larger potential winnings.

We will also use DF to find frequent itemsets. In order to use DF we need our data in a different format. We have to decide on a number of attributes that can either be true or false for a given record. These attributes are numbered starting at 1. The first line of the input file should contain the number of records. After that, input file should have a line for every record, containing the numbers of the attributes that the record has. For example, the line

1 3 4 7 12

means that there is a record that has attributes 1, 3, 4, 7 and 12, but none of the other attributes. DF will determine frequent itemsets, which are combinations of attributes that occur together in a record at least a certain number of times.

For the actual data mining we've limited ourselves to the data for January 1996. The database contains information on 30,557 hands for this month. Our script produces 275,477 records for this data. We will try to use all of this data where possible, but in some cases we may have to use only part of the data due to time constraints.

7.1 Basic statistics

First we'll look for some basic statistics that may be interesting. We don't use the Weka data mining suite because we can get most of these statistics using just MySQL.

On average there are roughly 5.9 players per hand. The number of players this month varies from 2 to 23. This is quite a bit more than in regular Texas Hold'em games, but most hands are played with a smaller number of players. Just a little over 11% of the hands was played by over 10 people and a little over 5% of the hands was played by more than 12 players. Still, these hands with a lot of players could have a notable effect on the amount of folds for example, since hands with more players are harder to win and make medium strength hands less playable.

The following tables show the frequency of actions per phase.

Pre-flop	
Action	Frequency
All-in	3 %
Bet	0 %
Call	20 %
Fold	63 %
Check	5 %
Raise	9 %

Flop	
Action	frequency
All-in	5 %
Bet	19 %
Call	11 %
Fold	25 %
Check	37 %
Raise	3 %

Turn	
Action	frequency
All-in	5 %
Bet	19 %
Call	11 %
Fold	17 %
Check	46 %
Raise	2 %

River	
Action	frequency
All-in	4 %
Bet	19 %
Call	9 %
Fold	16 %
Check	50 %
Raise	2 %

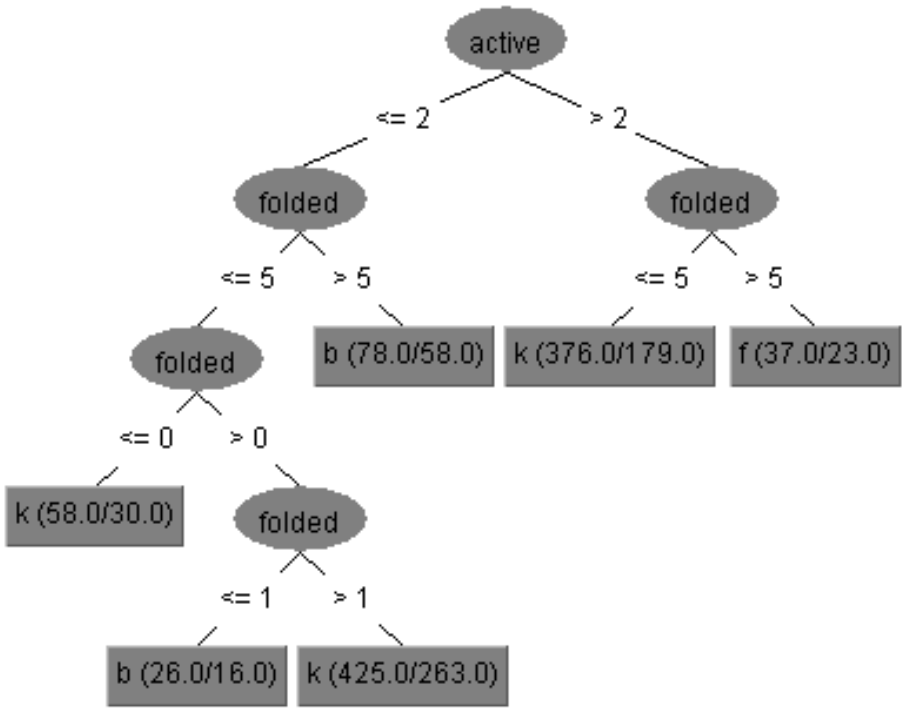
There aren't really any surprises here. Before the flop there are no bets because before the players get to act two players have to put in the blinds. When the first player gets to act there is already something in the stack, so players who want to play their hand can only raise the amount, call or go all-in. The amount of checks before the flop is very low because the only chance

to check is for the big blind if none of the other players raised or went all-in. The percentage of folds goes down as the hand progresses, which makes sense: people who decide to play their hand through a number of phases apparently have some amount of confidence in the strength of their hand.

7.2 Behaviour in relation to number of opponents

The number of opponents is an important consideration when deciding whether or not to play a hand. A mediocre hand can be playable if most of your opponents have already folded, while a somewhat strong hand can be a risky play against a large number of opponents. In trying to find some relation between the number of opponents and a player's behaviour we only look at three columns: the number of active opponents, the number of inactive opponents (who have folded) and the player's action.

With 41.5% of all cases, checking is the most common action in our data set. This means we can predict 41.5% of all cases correctly by always predicting a check. Using the J48 algorithm [17] we were able to consistently get slightly better results than choosing the most common option. The average performance of 42.1% correct predictions isn't very exciting considering the mere 0.6 percentage point improvement over simply always predicting a check. However, it is a slight improvement and in the decision trees we can see some interesting splits, as well as some less interesting ones. We will look more closely at one of the decision trees generated in Weka.



On the first node the tree splits in two on whether there are more than two active players left. Considering that one automatically wins when there are no opponents left, the tree effectively splits into a one-on-one part and a part for multiple opponents. In the case of multiple opponents, when more than five opponents have folded the most common action (making up almost two thirds of all cases) is to fold. This may be because one of the first players to act raised by a significant amount and all the other players are giving up because of this. However, when five or less opponents have folded the most common action is to check. Checking is the most common action in general, but in almost half of these cases the player chose to check. This is more than the over-all average, though not by much. One possible explanation is that when there are so many players in the game it is very tempting to check just to see how the flop turns out. If one player raises strongly this isn't as tempting, but if nobody raises most players will consider just checking and deciding later on whether or not to play on.

The other part of the tree deals with one-on-one situations and for the most part does not provide interesting information. There are two points worth mentioning. Firstly, if there is only one active opponent left and over five players have folded, the most common action by far is to bet. This makes sense: the player has a chance to steal the stack with only one player left to challenge him. However, that this action is so popular in this situation, but far less popular with less folded opponents is somewhat surprising. It may well be because against such a large number of opponents one has to make the most of every chance to win some chips. But the popularity of this move means that an observant player who is familiar with this habit may be able to exploit it. A bet in this position says little or nothing about the player's hand and it is a nice opportunity to play a somewhat strong hand. Secondly, when at most one opponent has folded the most common action is also to bet. This is to be expected. It is safer to bluff on hands with only two or three players at the table. When there are eight people with the possibility of calling your bluff you need to be very careful. But against one or two persons, even if they call your bluff you may still have a good chance of winning.

7.3 Cards in hand

We want to look for a relation between the value of cards in hand and whether those cards are suited.

We only have data on cards in hand for hands that players choose to play to the end. This can be rather limiting when it comes to data mining, but we can investigate these hands to determine what hands people generally choose to play. We choose the following characteristics to consider:

1. First card in hand is an ace.
2. First card in hand is a king.
3. First card in hand is a queen.

4. First card in hand is lower than a queen.

5. Second card in hand is an ace.

6. Second card in hand is a king.

7. Second card in hand is a queen.

8. Second card in hand is lower than a queen.

9. Cards are of the same suit.

We divide the value of the cards into four groups: ace, king, queen and lower. We do not make a distinction between lower value cards because we consider them to be "weak". This is a somewhat arbitrary decision, but it is important because having fewer characteristics may lead to more understandable output. If a player chooses to play a hand with $6\diamondsuit A\diamondsuit$ to the showdown, that results in the following line for our DF input:

4 5 9

This is because the first card is lower than a queen, the second card is an ace and the cards are suited. A similar line is inserted for each pair of cards that is included in our database. DF will read all these lines and look for combinations of attributes that occur frequently. We only look at records with data on the player's hand. Our 66,311 line input produces the following output when we run DF with a minimum support of 6000:

```
(66311)
7 (6961)
3 (7087)
2 (7761)
6 (8077)
5 (10259)
5 4 (6773)
1 (10304)
1 8 (6753)
9 (21562)
9 8 (13899)
9 8 4 (8147)
9 4 (13828)
8 (41013)
8 4 (26316)
4 (41158)
```

The first number between parentheses (66,311) indicates the number of input lines. Each of the following lines describes a frequent itemset and its support. We see that the itemset {7} has a support value of 6,961, meaning it is a subset of 6,961 of the 66,311 input lines. Looking at our list of characteristics above we see that this means that 6,961 hands had a queen as second card. We see quite a few frequent itemsets consisting of one item. Generally speaking smaller

itemsets occur more often than larger ones. Each occurrence of an itemset also counts as an occurrence of each of its subsets. For example, each itemset which has {1, 8} as a subset also has both {1} and {8} as a subset. This is the so-called Apriori-property on which most frequent itemsetalgorithms are based. The itemsets consisting of a single item simply indicate how often each of our attributes occurs. We see, for example, that 10,304 revealed hands had an ace as their first card, and 10,259 hands had an ace as their second card. We see that the most common cards in the revealed hands are lower than a queen. Aces also occur very frequently in these hands. And 21,562 hands were suited.

The most common itemset with more than one item is {4, 8}, which means both cards are lower than a queen. Over a third of the hands that were eventually revealed were made up of two cards lower than queens. This is more than we expected. Given that {4} occurs 41,158 times out of 66,311 and {8} occurs 41,013 times out of 66,311, the expected frequency of {4, 8} is:

$$\frac{41158}{66311} \times \frac{41013}{66311} = \frac{25456}{66311} = 38.4\%$$

The actual occurrence rate is $\frac{26316}{66311} = 39.6\%$. This type of hand makes up the majority of the hands players receive, but a hand like this is usually not very strong. We didn't expect this many to reach the showdown, or even win. The expected frequency of 38.4% does not take into account that, to a poker player, there is a huge difference between a hand with one high card and a hand with no high cards.

Some of the low hands probably made a pair. For example, two jacks make a fairly strong hand. But most hands with two somewhat low cards, even low pairs, aren't very strong. A pair of fours for example, is often not enough to win a hand. This is especially true when there is a large number of opponents, each of whom has a chance of beating your hand. It's possible that players sometimes play a weak hand to the end because they have some reason to think their opponents have even weaker hands. Or perhaps because they have already invested in their hand and simply want to see the showdown so their weak hand may become stronger.

The most interesting way to make a strong hand with low cards is to draw a flush. A flush requires five cards of the same suit, but does not require any high cards (though high cards in a flush make it stronger against other flush draws). Out of 26,316 low card hands 8,147 were suited. These hands had a decent chance at a flush draw, making them stronger than similar unsuited hands. However, the ratio between suited and unsuited hands is roughly equal for hands with cards and hands with high cards. Apparently people are willing to play low card hands quite often, regardless of whether or not their cards are suited.

7.4 Pot and stack sizes

We also want to look at the effect of a player's stack size and the size of the pot on a player's action. It could be that certain actions occur more frequently than others when the pot is very large compared to the size of the player's stack.

Once again we use DS to find frequent itemsets. To investigate this we chose the following characteristics:

1. The pot is larger than or equal to the player's stack.
2. The pot is smaller than the player's stack.
3. The player folds.
4. The player checks or calls.
5. The player raises or bets.
6. The player goes all-in.

In this case we look at all records, not just those with data on the player's hand. An example input line for a player who chooses to call, while his stack has 200 chips and there are 200 chips in the pot:

1 4

When we feed our 275,478 line input to DF with a low minimum support (973 or less) we get the following output, which lists all itemsets:

```
(275478)
1 (8596)
1 6 (1891)
1 5 (1806)
1 4 (3925)
1 3 (973)
6 (9899)
6 2 (8008)
5 (36260)
5 2 (34454)
4 (91300)
4 2 (87375)
3 (137999)
3 2 (137026)
2 (266881)
```

We find that in roughly one out of thirty-two cases the pot started out larger than the player's stack. We have to keep in mind that we only know the pot size at the start of the phase. Cases in which the pot increased to a size larger than that of a player's stack by an earlier player's action cannot be recognized in our database and are grouped with cases in which the player's stack is larger. We also see that folding is more common than checking or calling, which in turn is more common than raising or betting. Going all-in is the least common action.

It is not very surprising that we find very few cases in which the pot is larger than the active player's stack where the player chooses to fold. Apparently players don't often feel like passing up on such large potential winnings. Given that {1} occurs 8,596 times out of 275,478 and {3} occurs 137,999 times out of 275,478, the expected frequency of {1, 3} is:

$$\frac{8596}{275478} \times \frac{137999}{275478} = \frac{4306}{275478} = 1.56\%$$

The actual occurrence rate is $\frac{973}{275478} = 0.35\%$. This low rate of folds when the potsize is larger than the player's hand is also reflected in a higher rate of all the other actions. Most notably all-ins are much more common. This was to be expected. When the pot is very large a player will have to put in a lot of chips to keep playing. When the required number of chips is equal to or larger than the player's stack the player must go all-in.

7.5 Frequent itemsets in broader input

Finally we will look for frequent itemsets in input that consists of a larger number of attributes, which do not necessarily expect to be related. We chose the following attributes:

1. The current phase is pre-flop.
2. The current phase is flop.
3. The current phase is turn.
4. The current phase is river.
5. The pot is larger than or equal to half the player's stack.
6. The pot is larger than or equal to the player's stack
7. There are more active players than inactive (folded) players.
8. There has been at least one all-in.
9. The player folds.
10. The player checks or calls.
11. The player bets or raises.
12. The player goes all-in.
13. The player makes a profit on this hand.

Given these attributes, one possible input line would be

2 5 12 13

which means that the flop has been revealed, there have been no all-ins and there are more inactive than active players. The pot has fewer chips than the current player's stack, but at least half as many. The player decides to go all-in and ends up making a profit this hand.

We feed our 275,478 line input to DF with a minimum support of 10,000. If we ignore all single item itemsets we get the following output:

```
(275478)
3 10 (12579)
5 2 (12091)
5 10 (10842)
11 2 (11775)
11 13 (25926)
11 13 7 (12336)
11 13 1 (11979)
11 7 (17700)
11 7 1 (13421)
11 1 (17277)
2 13 (17017)
2 10 (26060)
2 9 (13622)
2 7 (13285)
13 10 (28475)
13 10 7 (11532)
13 10 1 (14113)
13 7 (25165)
13 7 1 (18583)
13 1 (28888)
10 7 (38013)
10 7 1 (26960)
10 1 (45175)
9 7 (94002)
9 7 1 (90507)
9 1 (118538)
7 1 (133049)
```

These are the frequent itemsets with the highest support, excluding trivial itemsets which only indicate that a given attribute occurs at least 10,000 times. We see certain attributes appear in more frequent itemsets than others. For example, 1 and 7 are very common, while we don't see 4 in any of the frequent itemsets. This is because certain attributes occur more often than others, and are thus more likely to combine with other attributes to form a frequent itemset. The large number of actions during the pre-flop compared to other phases means frequent itemsets are more likely to have attribute 1 than 2, 3 or 4. Attribute 13 (the player makes a profit) is rather common because, in order to make a profit, a player has to take part throughout the entire hand, contributing more records than players who fold quickly.

Most of the frequent itemsets we see are made up of some combination of attributes 1, 7, 10, 11 and 13. Note that 10 and 11 never occur together because they each specify a certain type of actions, which can't be combined with another type of action in a single record. If we look at itemsets with lower support we see the same trend. Combinations of frequent attributes occur frequently, while combinations of less frequency attributes occur less frequently.

The only interesting exception we see in the above output is that {11, 13} (the player raises/bets and makes a profit) occurs almost as often as {10, 13}

(the player calls/checks and makes a profit), while calling and checking are generally more common than raising and betting. This might be because player with strong hands tend to raise or bet more often and also tend to win more often, while players with mediocre hands tend to call and check more often, even if they end up making a profit. It might also be because raising and betting actually increases your chances of winning, by sending a message to other players who are then more likely to fold.

8 Conclusions

In this paper we have described the pokerbot that we created for experiments on basic poker opponents. We have underestimated the amount of work that this takes and had to make a few cutbacks from the original plans. This led to the simplification of the opponents. We have finished with a pokerbot who will win against basic opponents but would be no match against a more sophisticated opponent. It performed better than expected: in one of the experiments it could win over 80% of the time.

Some unexpected results came from a basic bot who would never play a hand but still would last longer than half of the opponents. And even in tournaments with actual people this would hold up.

There is still a lot of room for improvement left. The most important item that is not done so far is that our bot is mostly concerned about his own cards and it should also be analysing its opponents. There is no opponent modelling in the project yet, and opponent modelling is exactly what is needed to create a decent poker playing bot. Unfortunately this would take a huge amount of work and creating a pokerbot is not something that can just be done in such a small project. When looking at other pokerbots there is often years of work and an entire team behind it.

In the data mining part of the project the results have been somewhat disappointing. It appears the data given provides rather limited possibilities for data mining and the results are not very useful. For future work it would be very useful to start building data from scratch and find a way to include all data on player's hands. This is not a simple task as this information should be kept secret from the other players at all cost. The only way we see right now is to write a poker server that incorporates this feature. One might think that it is enough to keep the information secret until the hand is played, but knowing how a player handled certain cards in the past can be extremely useful when playing that player again. Even making the data available a few days after the game has been played (which in itself would be difficult to achieve) would be considered inappropriate by most poker players. The problem with writing a poker server is that you also need to attract people who are willing to play a considerable number of games of quality poker.

A lot of on line poker players use data mining suites specifically designed for poker to improve their play, but these are geared towards keeping track of how often a player plays and under what conditions he or she folds. This is the kind of

information that professional poker players keep track of themselves during play and which does not really open up new possibilities for a computerized poker player. However, it may be that this basic tracking is the limit of the extent to which data mining will be successfully applied to poker. For pokerbots it may be more profitable to rely mostly on statistics and to perhaps use basic machine learning (neural nets for example) to try to classify opponents into groups like "loose" and "solid".

9 Glossary

All-in: A bet or raise of all the chips in front of you. If someone moves all-in he puts all his chips in the pot.

ante's: Money placed in the pot by all players at the beginning of a hand. A variation of the blinds that we use, in Multi-Table-Tournaments blinds and ante's are often combined after a few rounds.

Big Blind: A force bet made by the player to the left of the small blind, to generate the action.

Big stack: The player with the most chips at the table.

Blinded away: If the player with the short stack doesn't play many pots, he may lose all his chips when it is his turn to pay for the blinds.

Button: The player to the right of the small blind, who represents the dealer of that hand. He gets to act last after each betting round after the flop. To be on the button is a great advantage.

Bully: A player with a big stack who tries to push the other players around.

Flop: Three cards turned face up simultaneously in the centre of the table. These are the community cards that each player can use. The flop is followed by a round of betting.

Hand: Each time you get dealt two hole cards, it is the time you have a hand. Sometimes it refers to one complete deal or game. The intended meaning of "hand" should be clear from the context.

Hole cards: The two down cards dealt to each player at the beginning of the hand. No other player can see these cards. These are also called your starting hand.

Nuts: The player with the best possible hands has the nuts. Pre-flop is AA the nuts.

Out: A card which will give a winning hand if it arrives. If you have 4 cards to a heart flush you have 9 outs (9 heart cards left in the deck) to hit the flush.

Pre-flop: The action before the flop is dealt.

Premium hands: Very good starting hands like AA or KK.

River: The fifth and last community card that is put in the centre of the table face up. Followed by the final betting round.

Short stack: The player with the fewest chips at the table.

Showdown: After the final betting round the people who are still in the hand show their hands to determine the winner.

Small blind: A forced bet made by the player to the left of the dealer/button to generate action.

Tightness: How tight an opponent is, a very tight player plays less then 10 % of the hands he gets dealt. A loose opponent will play more then 50 % of the hands he gets dealt.

Turn: The fourth community card that is put in the centre of the table face up. Also followed by a betting round.

References

[1] Doyle Brunson. *Syper System*. Cardoza Publishing, third edition, 1978.

[2] Python community. Python programming language — official website. `http://www.python.org`.

[3] Andy Dustman. Sourceforge.net: MySQL for Python. Website. `http://sourceforge.net/projects/mysql-python`.

[4] D. Harrington. *Harrington on Hold 'em, Volume 1: Strategic Play*. Two Plus Two Publishing, second edition, 2004.

[5] Texas Hold'em. Wikipedia. Website. `http://en.wikipedia.org/wiki/Texas_hold-em`.

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

[7] Walter Kosters. DF. Website. `http://www.liacs.nl/home/kosters/df/`.

[8] H.W. Kuhn. A simplified two-person poker. Contributions to the Theory of Games. pages 92–103, 1950.

[9] Michael Maurer. Irc poker database. Website. `http://games.cs.ualberta.ca/poker/IRC`.

[10] J.F. Nash and L.S. Shapley. A simple three-person poker game. Contributions to the Theory of Games. 1950. Princeton University Press.

[11] The University of Alberta. The University of Alberta computer poker research group. Website. `http://games.cs.ualberta.ca/poker/`.

[12] The University of Waikato. Weka 3 — Data mining with open source machine learning software in java. Website. `http://www.cs.waikato.ac.nz/ml/weka/`.

[13] Wim Pijls and Walter Kosters. Apriori: A depth first implementation. First Workshop on Frequent Itemset Mining Algorithms. 2003. Online proceedings `http://www.liacs.nl/~kosters/dffinal.pdf`.

[14] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, second edition, 2003.

[15] Kevin Suffecool. Cactus Kev's poker hand evaluator. Website. `http://www.suffecool.net/poker/evaluator.html`.

[16] J. von Neumann and O. Morgenstern. *The Theory of Games and Economic Behavior*. Princeton University Press, second edition, 1947.

[17] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, second edition, 2005.