Probabilistic beliefs in poker

Using math and beliefs to predict hole cards

19-4-2011

(7,5 EC)

Author:
Jori van Schijndel
ID: 3345297

Supervisor:
Jan Broersen (UU)

Preface

The last part of my bachelor education is the writing of a thesis. The goal of this thesis is for the student to do an individual project with only the help of a supervisor. It is excepted that the depth and complexity of the project reflect that level which would be expected from an academic student.

It is of course expected that the subject has a sufficient linking to CKI.

This project will hopefully be my last before getting my bachelor degree. After two and a half years CKI the time came to write this thesis. Together with my supervisor – Jan Broersen – we came up with the subject of poker AI.

Because a full research project of the field of poker AI would only give time to scratch the surface, the choice is made to pay special attention to the prediction of the opponents cards together with the statistical analyses needed for an poker agent to make a decision.

Due to the scope of this work, this thesis isn't nearly as comprehensive as I would like it to be: Many interesting and relevant topics are out of the scope of this project, and are thus not discussed.

Special thanks to Jan Broersen for helping me with this project.

Jori van Schijndel
ID: 3345297

Abstract

Games are an interesting topic for computing science. Poker is especially interesting to the field of AI. It gives an ideal working ground to test and experiment with different topics from heuristics to deliberate misinformation.

In this paper we went through the process of designing an AI poker agent. There is an overview of both the possibilities as well as the challenges faced. The main focus is on the prediction of the opponents cards.

Writing a complete poker agent and poker program – for interaction between agent an human – is next to being technical challenging also very time consuming. Because of this some aspects from a poker player are only explained theoretical and are not implemented.

But during this process a working prototype of an AI poker agent as well as an poker game is build.

TABLE OF CONTENTS

Chapter 1

Introduction

Playing games comes natural to most people. As a child you play games with peers and your parents, you design your own (made-up) games, you win and lose, and you learn. The structure there is in games - goals, rules, skills and strategy – makes for an ideal playground to prepare for real-life as sort of a simulation. This same structure makes them also interesting for computing science and AI.

1.1 Study of games

The study of (mathematical) games of strategy is interesting to computer science and AI because of their general properties:

- Games have well-defined rules
  - This makes them easy to implement. It is almost trivial to build an agent capable of playing a 'legal game'.
- Games have complex strategies
  - Most puzzles belong to NP or PSPACE[1]. Learning to play those games automatic – perhaps with good heuristics - is learning to create (mathematical) cleverness.
- Games have specific (end) goals
  - Which makes it easy to quantify success and gain.
- Games resemble various "real life" activities.
  - Games can be used to learn(about) or to simulate those activities.

[1] M. R. Garey and D. S. Johnson, Computers and Intractibility: a guide to the theory of NP-completeness, W.H. Freeman, 1979

1.2 poker

The popularity of poker has skyrocketed since the beginning of the 20th century. It has changed from being just a recreational activity to an international spectator sport. Perhaps story's of unknown amateur players winning big money at tournaments are responsible for this gain in popularity.[2]

But what aspects of this game are interesting to the academic world and more precise to the field of computer science and AI?

The game of poker is an ideal testing ground for automated reasoning in an uncertain world. Not only is it interesting because it is a zero-sum[3] game of incomplete information, but also because of the randomization of the cards, the unpredictability of opponents, strategy exploits, the possible cooperation between players against a common opponent, and the sequential character of the game which requires a complex and adaptive strategy. Making it more than just a game of chance.

For poker the rules of the game are quite simple, yet to play the game no simple approach will suffice. The design of a computer agent will be no simple task.

although the field of AI has shown its capabilities in designing adaptive agents to play games like chess and checkers (and more recently Jeopardy[4]) at the level of skilled human players and beyond, for the game of poker no such agent has yet been designed, capable of defeating the world's best players consecutively .

1.3 Thesis subject – Aims and objectives

The goal of this paper is to look more closely to a specific part of a theoretical poker agent: the part responsible for predicting the opponents cards and thus knowing the changes of winning or losing. This information is vital for the poker agent to make a strong decision.

The prediction of the opponents cards can be handled purely statistical, using only the information given by the cards observable to the agent. Another and more favorable way is to use extra information about the game, poker players in general, and the opponent to make a possibly more accurate guess. Perhaps in the same way a real-live poker player try's to predict his opponents cards by studying him.

During a game of poker and over a series of games increasingly more information is revealed. Information about the moves made by a player can be used for opponent modeling.[5] With a good model of the opponent his actions will convey information about his cards. This can be combined with the statistical information given by the cards. During each action in the game information is revealed and so guesses about his cards can be adjusted accordingly. this improving (or updating at least) of the guesses can be done with the arithmetic's like Bayes's Theorem.

---

[2] http://www.articlesbase.com/online-gambling-articles/poker-popularity-on-the-rise-1270067.html
[3] Ignoring the possible house-cut. probably games like strip poker also lack this property.
[4] http://www-03.ibm.com/innovation/us/watson/index.html
[5] predicting opponents play and character.

1.4 thesis outline and focus

This report will try to outline the challenges faced when designing such an AI poker-agent and will look more specifically at the design considerations faced when constructing a system or architecture that is capable to predict to a certain degree of accuracy the (hole) cards that are present on the table during a game of heads up (two –player) poker. Also close attention will be paid to the mathematical part of such a system. Combining old probability estimates with new information will play a major role in the mathematical part of this paper.

The development of a poker AI is presented throughout the paper in a purely theoretical fashion. the designing, building and testing of a complete agent would go beyond the goal of this (theoretical) paper and beyond the time set (7,5 EC). Some parts of the agent will only be explained in terms of an architecture while other parts will be explained in terms of pseudo-code or working code.

Chapter 2

Poker and it's forms

Purely for practical reasons a (simple) explanation of the various games that belong to poker is needed. Next to Texas Hold'em – the most played form of poker – also a more limited and simple form is explained that is implemented for this thesis.

2.1 General Rules

Poker is not just one card game, but a family of card games. The difference between the various games is the difference in rules. There is variation in the betting-rules, the dealing of the cards, the value of different hands and so on.

The (general) equality between the games is that the games begin with some sort of forced bet – to make sure the pot is never empty – followed by voluntary betting . the players need to match each other's bet to stay in the game. The players have their own private cards and in some games – like Texas Hold'em – there are also community cards. The value of a hand in community card poker (also known as flop poker) is determent by the best possible combination given the community cards and the players private cards. With no community cards the value of a hand is purely determent by the private cards. A player wins if all other players fold or when the game reaches a showdown and his hand is the strongest still in the game.

The most known groups of poker:

- Stud poker
- Draw poker
- Community card poker

The most known variations of poker:

- Texas Hold'em
- Omaha
- 7-Card stud
- 5-Card stud
- Triple Draw
- Crazy Pineapple
- Draw
- Razz

2.2 Texas Hold'em

In Texas Hold'em each player is dealt two private cards – known as Hole cards – which value is only known to him. Five community cards are dealt to form the board and are known and available to each player. The players use these shared cards together with its own cards to form the best possible five card hand. The player can use zero, one or two of its private hole cards to form his hand.

2.2.1 Poker hand ranks

The value of a hand is determent by de poker hand ranks. Hold'em uses 'high' poker rankings:[1] (from high to low)

- Straight Flush : Five cards in sequence of the same suit.
- Four of a kind : Four cards of the same rank. Followed by a kicker.
- Full house : Three cards of the same rank and two cards of a different, matching rank.
- Flush : Five cards of the same suit.
- Straight : Five cards in sequence.
- Three of a kind : Three cards of the same rank, and two unrelated side cards.
- Two pair : Two cards of the same rank and two cards of a different, matching rank and a side card.
- Pair : Two cards of the same rank and three unrelated side cards.
- High card : Any hand that doesn't qualify under a definitions listed above.

2.2.2 Variations of Texas Hold'em

- Limit Texas Hold'em : There is a pre-determined betting limit on each round of betting.
- No limit Texas Hold'em : A player can bet any amount. Perhaps limited by the size of the pot.
- Pot limit Texas Hold'em : A player can bet up to the amount in the Pot.
- Mixed Texas Hold'em : during the games there is switched between limit and no limit.

A round of Hold'em consists of the dealing of the hole cards to the players which is followed by two forced bets (and possibly by addition small forced bets by all players known as antes). After that the players have the possibility to fold, call of raise. When everyone has either folded or matched the amount put in by all other active players the 'pre-flop' round ends and is followed by the flop: three cards are placed face-up on the board and is followed by an additional betting round. After that round another card is placed face-up (turn) again followed by a betting round. The last community card is placed on the board during the river. This is again followed by betting. If this round ended with two or more players the hand ranks of the different players is compared and the player of players with the highest hand get or split the pot.

2.2.3 Relevant poker terms

- Poker rounds: pre-flop, flop, turn, river, showdown
- Poker moves: check, call, raise, bet, flop

[1] http://www.pokerstars.com/poker/games/rules/hand-rankings/

2.3 Own version

In this thesis the implementation is limited to a simplified version of Texas hold'em. This is because of practical limitations and that a full implementation of a poker agent goes beyond the goal of this paper: most ideas outlined in the rest of this paper can also be applied the real world Texas Hold'em.

Because the workings of a game is determined by its rules, a listing of the applied rules will suffice to make a (simple) poker program for the AI agent and human player to interact with.

2.3.1 Rules

We use a variation of limit heads-up poker.

- **Blinds**: the dealer puts in the small blind ($5) and the player the big blind ($10).
- **Round**
  - Preflop : Both players are dealt two hole cards
  - Flop : three cards are placed on the board
  - Turn : one additional card is placed on the board
  - River : one final card is placed on the board
  - Showdown : players hole-cards are revealed to determine winner
- **Player order**: the dealer begins in the first round. The player in the other rounds
- **Betting:** bets are of fixed size. $10 during preflop and flop and $20 during river and showdown. During the first two round there is a max of one bet and two raises and during the turn and river one additional bet is allowed.
- **Winning:** A player wins if the other player folds or he has the stronger hand during the showdown. With an equal hand the pot is split. Hand ranks are the same as discussed in 1.2.1.

2.3.2 layout (GUI) and architecture

2.3.2.1 GUI

The user interface consists of two distinct parts: the board and controller.

On the board the community cards are displayed (face-down or face-up depending on the stage of the game) and the hole-cards of the two players. (these are placed face-up for convenience, but this can easily be changed to display only the players own cards face-up.) the last section of the board displays the beliefs the AI has over the possible final hands the player and it will end up with, the chances of winning and the probability of the opponents hand falling into the different groups (discussed in chapter 3.2.1). He uses this information to determine his own next move.

The controller is the interaction of the player to the board. It displays the legitimate moves the player can make, the distribution of chips between the players and the pot and an overview of the actions that are taken by the players.

Images of both board and controller are present in the image overview at the end of the paper.

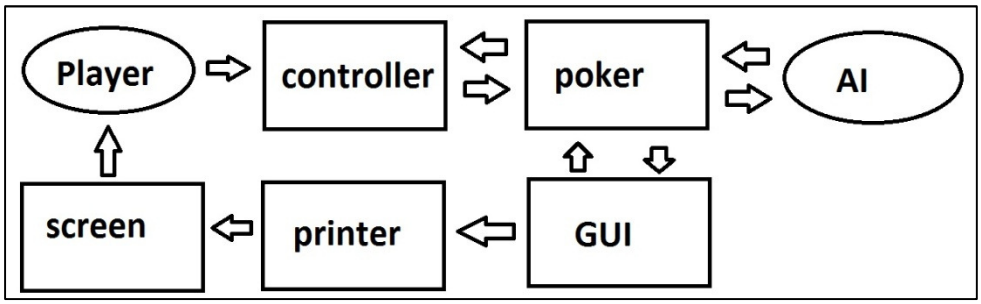
2.3.2.2 architecture

The architecture used and explained below is divided into two parts: the poker-game and the poker-player. The first part is responsible for the GUI, the rules of the game, dealing of cards and so on. It contains two players. One threw the controller – player – and the other is the poker-player.

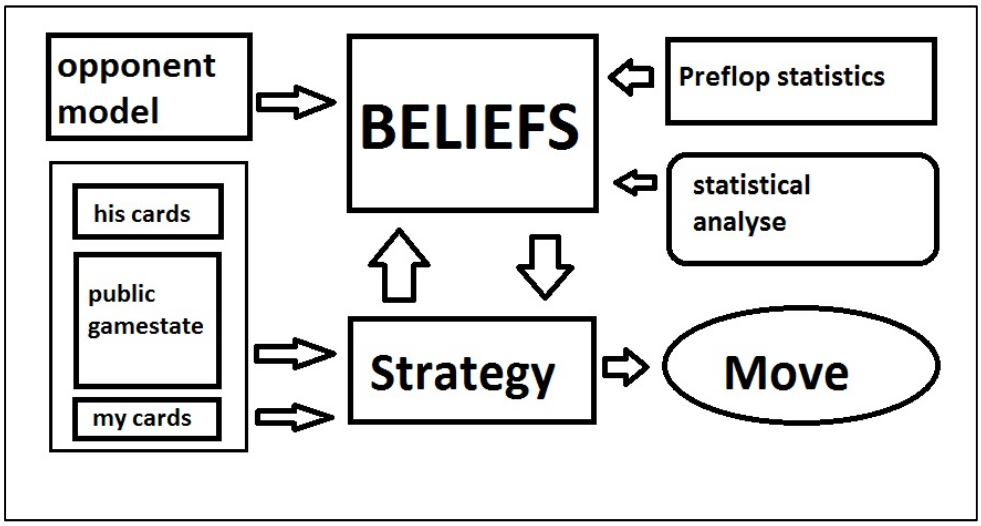
The poker-player interacts with the game in a similar matter as the human player – you are presented with the public and private game state and a series of legitimate moves, and your return type is one of those moves – but doesn't use the controller.

The architecture of the poker player consists of a strategic part, a beliefs part, an opponent model and statistical modules.

2.3.2.3 poker-game architecture



2.3.3.4 poker-player architecture



Chapter 3

Simple opponent model

Although opponent modeling isn't part of this paper, it is an essential part of our previous explained architecture. So a simple introduction to this topic is needed as well as a not very effective but working version of a 'opponent model' for our AI agent to function.

3.1 General ideas

3.1.1 The need for an opponent model

when playing a game it is often assumed that the opponent has a similar goal [1]and uses some sort of a similar strategy. This idée is used in the commonly applied search algorithm mini-max. when using this idée in for example poker you are not able to exploit the faults or weaknesses of your opponent or his strategy.

This is because this method does not use any knowledge of the actual opponent. It does not discrimate between players or changes over time when more information about the player is accessible.

Opponent modeling is a method used to incorporate this knowledge of the opponent. Another advantage of an opponent model is that the heuristics used in this method, do not require a full search of the game three. Because of the complexity of poker, this is computational more effective.

This part of the program is needed for the prediction of hole cards if we want it to perform (potentially) better than pure probability.

[1] Similar, though opposite

3.1.2 Differences between opponent models

There are different ways for such a model to work, but there are three to four main groups:

1) probabilistic models
2) neural network models
3) rule-based
4) (pre defined models)
    - This model can be in the form of any of the three above, but differ in that they do not adjust during play but that the model assigned to the opponent changes .

Of course it is also possible to use a combination of these groups. For instance the 'hybrid' form used in one of the most known AI poker programs 'poki' incorporates an meta-predictor that uses the information from different opponent-modeling techniques

3.1.3 workings

Although the implementation details differ between games and designs the basic idée is that (relevant) information is passed on to the model during play to tune the model. This can happen online, or batch like or in another way. Than during play the computer can use that model to predict certain actions or for instance the hole-card the opponent has.

3.2 our opponent model

because the opponent model used in our system is used because of its practical necessity, a very simple static[2] rule-based model is used. In the back at the 'code snipits' a simple version is visible.

3.2.1 workings

The idée in this implementations is that a given action of the opponent changes the probability he has of holding certain hole-cards[3]. If the opponent bets or raises, he probably has better cards than when he check's or call's. So the actions that signal for good cards increase the change of him holding good cards en decrease the change of him holding bad cards and vice versa.

3.2.2 improvements

It is easy to see fault in this design/idée. We do not incorporate the fact that a player can bluff, that he is slow playing, the pot-value, the Preferability of certain groups given known cards and so one, but that al falls out of the scope of this project.

[2] Doesn't change over time and uses a simple pre-defined model of a poker player's strategy
[3] All possible hole cards are divided into 9 groups. We use the ranking proposed by author and poker player David Sklansky: "david sklansky & mason malmuth – Hold'em poker for advanced players"

Chapter 4

Strategy

At various points in the game the agent needs to make a decision. How he does that can be called his strategy. The decision to fold, check, call, raise or bet requires an analyses of the situation – the mathematical part – in combination with consideration of his goals. Next to winning, his (sub)goals can be deception (hiding his cards and strategy) or exploration (trying to access the opponents cards and strategy).

4.1 General ideas

Perhaps the most important part of a real poker program would be the strategy in terms of succes. Although it is possible to handle the game purely mathematical and use a function that gives an expected value for a situation and handle directly according to that –if EV() < 1 stop, if EV()>1 stay in the game or if EV()==1 do a random action. – but that would make you a mathematician and not a poker player.

Given the analyses of the situation, the agent needs to make a smart decision. Normally given a high chance of winning the player wants to stay in the game as to leaving it when his chances are slim. But pot-odds, bluffing, aggressive play, deliberate misinformation and so on make it not as trivial as that.

4.2 Our implementation

Because the strategic part is only used for practical reasons, we make use of a very simple idée. When the expected change is above 85% bet or raise[1], if the chance if between 35 and 85% check or call and below that chance fold.

- we do not incorporate pot-odds. When the amount needed to stay in the game is low in comparison to what is in the pot, folding would be less likely
- We do not incorporate bluffing. When bluffing you essentially overestimate your own chances and act according to that.

[1] Or check/call if the limit on raises is met.

Chapter 5

Ranking the cards

When we are dealt our two hole cards we try to assess our chances of winning. When we think we have a decent chance of winning we want to stay in the game as to folding when our chances are low. There are different ways to make an educated guess towards your chances of winning and these different ways can be used in different times and they all have their pro's and con's.

5.1 General ideas

Given every information available the (AI) player will – in a normal situation – not be able to determine with certainty the cards the opponent has. But it is possible for it to have certain beliefs. Given his own hand, the cards already on the board, the opponents actions and the information given by the opponent model certain hole cards are more likely than others. These beliefs can be used to guess the change of winning. The 'strategy part' of the AI needs that information to determine its next action.

Because of the nature of the game the tactics used in our agent can be divided in to two different sections: the beginning and everything after that.

5.2 Begin

After we our dealt our two cards, we have no information next to our own two cards. Because of this our task is purely probability theory.

There are 52 possible cards and only the nine top cards are used[1]. Because of this there are (52!/43!)/ (2!*2!* 5!)[2] Possible ways to distribute the cards. But because in a lot of situations only the rank of a card is relevant , the real number of different combinations is lower. Given the two known hole cards the number of possible combinations is significantly lower (50!/43!)/(2!*5!)[3] .

Producing all these combinations and counting the percentage that the AI will win is a trivial task[4], but to make a search tree on-the-fly of all these possibility's and calculate the number of wins and losses will be to computationally intensive for an efficient AI agent.

So how do we go about designing an 'algorithm' or program to tackle this problem in an efficient way?

---

[1] This in the case of heads up poker without 'burning' the top cards.
[2] 2.781.381.002.400
[3] 2.097.572.400
[4] An 'pseudo-like' Haskell example is given in the back

Chapter 5: Ranking the cards

The input to the program can be of two types.:

- 2 cards: Hole and rank: 1326 possible combinations : (52 * 51)/ (1 * 2)
- 2 cards: rank and suited or not: 91 possible combinations : (13 * 12)/(1 * 2) + 13

We can see a tradeoff between information and efficiency. Because in most real-live poker situations the second notation type is used – Apparently that gives enough information – we will use that idée in our program.

Given the hole cards we want a percentage that gives us the changes of winning:

- Statistically: the changes of winning is determined by a search tree.
- Empirically: the changes of winning is determined by analysis of 'real live games'.

This is essentially an (non-injective) surjective bijection: $\forall_{hole_{cards}}(\exists_x(x \in \mathbb{R} \wedge f(holecards) == x))$
For every set of hole cards we want a function that gives us a real number . we want this number to reflect the changes of winning

5.2.1 Our implementation

For our implementation we will use pre determined changes[5].

For this we can use a static function in java. We use a two dimensional array to determine that chance for the non suited hole cards. the two ordinal values of the rank determine the location in the array. If the cards are suited the value is 3 points higher if the rank is two or three and else it is 4 points higher.

The code is presented at the back at the code snippets.

5.3 After the first action

After the first action of the opponent we can use the opponent model. Because of this our views on beliefs and chance changes.

We begin by assigning chances to having starting hands from the different groups discussed in 3.2.1. These changes are first purely statistical: each hand is equally likely, so the change of having starting hands of a groups equals the group size divided by the sum of all group sizes.

Each action is sent to the opponent model and that in turn changes the likeliness of the different groups. The likeliness of the different groups combined with the already known cards is used to calculate the probability of the different kinds of final hands and the chances of winning, losing or a tie for both the player and the AI.

The exact implementation of this system is explained in chapter 5.

[5] Statistics copied from: http://www.beatthefish.com/poker-strategy/texas-holdem-poker-hands.html

Chapter 6

Implementation

The statistical information needed by our agent needs to be calculated. There are of course different ways to do this. In most cases there will be a tradeoff between functions, efficiency and accuracy. In this chapter we will explain the choices I made in this implementation and give detailed information about the design and workings of the code

6.1 Choices

As said in the previous chapter, an accurate implementation can be achieved quite easily, but this method lacks efficiency. Because we want to use this module in an interactive program, calculation has to be relatively fast. So in our case efficiency is more important than accuracy.

We made this choice also because a very high level of accuracy isn't necessarily more reliable, because a large part of the input data – the likeliness of different starting hands – is also not very accurate.

The necessary function are (given the data) :

- Give the probability for the different possible finals hands of the opponent
- Give the probability for the different possible finals hands of the AI

Other functions could be:

- Give the probability for the different possible final hands, not using group probability
  - Is efficient if the group probability equals the starting group probability
    - This is the case in the beginning and if an opponent model isn't used.
- Give the chances of winning, losing or a tie.
  - This isn't implemented, because we made a function that estimates those changes using the probable final hands of the opponent and AI.

6.2 Design

6.2.1 Outside Haskell

The choice is made to program this part in Haskell instead of in java like the rest of the program. Haskell is a suited programming language for this task because it makes it easy to produce all the different combinations of hands, decks and cards needed.

Input: The input required is all known cards and likeliness of different groups of hole cards.
Output: The output is the change of getting the different types of hands.
poker_eval :: ([Int] -> [Int] ->[Int]) -> [Int] // pseudo-like
Because we want the graphic part of the program (java) to interact with the statistical part (Haskell), we need to take that into account. The statistical part is used as a static function, only the java part needs to call the Haskell part and not vice versa. Because of this the choice is made to compile the Haskell program to and exactable and let it be called by the java program throw the command prompt. This output is then parsed and used in the rest of the program.

6.2.2 Inside Haskell

The workings of the different functions is explained in the next subchapter, but here is an overview of the used function and their goal.

Main: The main of the Haskell program is of course of type IO(). The length of the arguments is tested to determine which function is called for.

Begin1: Give the probability for the different possible final hands, not using group probability
Begin2: Give the probability for the different possible finals hands of the AI
Begin3: Give the probability for the different possible finals hands of the opponent

Stap0, stap1, stap2,stap3: Route for begin1.
Stap0', stap1', stap2',stap3': Route for begin2.
Stap0'', stap1'', stap2'',stap3'': Route for begin3.

Extra Function:
filter' :: [Int] -> [[Int]] -> [[Int]]
filter: map over the second argument. If any of those ints are an element of the first argument, delete them.

fac :: (Num b, Integral a) => a -> b
fac : standard function for factorial.

ncrExtra :: [Int] -> [Int] ->[Int] ->[Int]
ncrExtra: give change if given number of tries, number of needed elements and length of set

ncr :: [Int] ->[Int] ->[Int]
ncr: give number of possible combinations

6.3 Workings

This chapter is to explain how we calculate the different chances needed for the program.

The basic idée is that we calculate the chance for every different hand rank. The Haskell program has a different lists (type :: [[Int]]) that contain every flush, straight, pair and so on. For every one of those list we calculate individually the chance of getting each possibility and take the sum.

We map over the list with every possibility and use a function that gives the change for each individual possibility given the known cards.

The possibility of an individual possibility is accessed in multiple steps:

1) Test if the possible situation can already be made given the known cards
    - If (true) then chance is 1 else step 2
2) Test if the possibility isn't possible. Not possible if opponent has one or more of the required cards or the number of extra cards required is less than the number of free places on the board.
    - If(true) then chance is 0 else step 3
3) Calculate the chance using the number of required cards from the deck, the deck size and the number of free places on the board. The chance is the number of correct combinations divided by the number of combinations that can still be made.

6.3.1 winning or losing

The simple tactics used by the agent uses only the estimated chances of winning or losing to determine its actions. This value is calculated by using the estimated final hands of both players.

the chance is of winning is – simplistically explained – the sum of the chance for each hand rank times the change for the opponent having a lower hand rank. For the situations of having equal hand rank there is a predetermined change of getting a tie[1]. in the situation of not getting a tie the chance of winning equals that of losing.

6.3.2 accuracy

As explained in 6.1 efficiency is more important than accuracy in our case. Although accessing chances as explained above can give a fairly accurate outcome, there are one major drawbacks.

When checking a possibility we do not exclude it, if with the same seven – board and hole cards – we can make a higher ranking card or a equally ranking card that we already included.

This means the chances are overestimated [2]. This problem would be off no concern if it overestimated with a constant factor, but that is not the case.

This is a tradeoff for which is chosen for.

---

[1] This depends on the rank. For example a tie isn't possible with a RoyalFlush.
[2] Or correct if these drawbacks aren't present, as is the case for the collection of RoyalFlush.

6.4 Further extensions / improvements

There are different ways to improve or chance the program with the goal to achieve better or more efficient play. Here are just a few possibilities.

- Full game tree search: if high accuracy is important.
  - This is trivial. But doing this fast is difficult. Perhaps a GPGPU solution would be efficient.
- Exclude possibilities for a lower hand rank.
  - Make an extra test if higher ranked hands are also possible
- Handel preflop estimates in Haskell instead of in java.
  - Except for implementation details this would be the same as in java
- Do a full search during endplay.
  - During endplay the possible different combinations is lower, so a full search can be done fast.
- Use tree search for winning/losing percentage.
  - Map over a full tree of distributions with a function that tells if you win, lose or reach a tie.
- Work for multiplayer games
  - Working would be the same. But data types need to be altered.
- Online working
  - The program is started all over for every call. Use a form of updating to include previous estimates to increase speed.

Chapter 7

Conclusion

Because of the practical approach in this thesis combined with a working prototype, the conclusion can be viewed as some last considerations together with some noticeable experienced I had during this project, than as an answer to a question.

7.1 Conclusion

As could be expected this agent isn't working well enough to play against perhaps even a novice player. But this is mainly because both the part of opponent modeling and strategy aren't implemented in even a mildly competitive manner.

The possibility's and considerations for both those parts are explained in few detail throughout this paper, but that is because the focus is laid on the math behind calculating the change of winning.

This part is essential in building a good poker player.

The different possibilities, choices and problems in calculating these chances is outlined in the previous chapters. The information there, should be sufficient to make a own implementation.

Although there are still some performance – both efficiency and accuracy – issues with our own implementation of the mathematical part of the agent, these can easily be overcome in a real implementation.

7.2 ongoing research

Further work on this project can go in different ways. Perhaps the two most interesting roads would on one hand be increasing efficiency and accuracy for the mathematical part and on the other the implementation of good working strategic and opponent modeling parts.

Other research can be in the direction of a deep analyses about the complexity of the game of poker, a game theoretical approach – think of nash equilibria – or a 'solution' to the game.

Coding examples

Haskell example of full search:

```haskell
-- code each card as an Integer
-- idee is that rank/suit are enum: than use enum -> ordinal and something like:
    -- value = rank.ordinal + suit.ordinal * 13
poker_program :: [Int] -> Double
poker_program myCards = fromIntegral(sum (map (\x -> win myCards x) possibilities)) / (fromIntegral(length possibilities) * 2.0)
    where
    possibilities :: [[Int]] -- list of al combinations of 7 cards you can make with the cards remaining
    possibilities = combinationsOf 7 ([1..52] \\ myCards)
    win :: [Int] -> [Int] -> [Int]      -- returns for a combinations of 9 cards:
    win my rest = WinOrLose my deck his             -- a score of 2 if he wins
        where                                       -- a score of 1 by draw
        deck = drop 2 rest                          -- a score of 0 if he loses
        his = take 2 rest
    winOrlose :: [Int] -> [Int] -> [Int] -> Int     -- give relevant score
    winOrlose my deck his = if ((score my deck) < (score his deck)) then 2 else (if ((score my deck) < (score his deck)) then 0 else 1)
    score :: [Int] -> [Int] -> Double               -- determine score by getting the highest value for each of possible 21 groups of 5
    score hole deck = maximum (map (\x -> score' x) (combinationsOf 5 cards))
        where
        cards = hole ++ deck
    score' :: [Int] -> Double   -- gives integer value according to type (flush/straight etc) and decimal value for kicker etc
    score' cards = .....        -- implementation not given

-- return list of al possible combinations
combinationsOf 0 _ = [[]]
combinationsOf _ [] = []
combinationsOf k (x:xs) = rev_map (x:) (combinationsOf (k-1) xs) ++ combinationsOf k xs
```

Java example of preflop change of winning:

cards.java | card.java | poker.java | controller.java | AI_poker_player.java | preflop.java | beliefs.java | simple_opponen

```java
public class preflop {

    private static int[][] not_suited= {
        {50,31,33,34,34,35,37,39,42,44,47,50,55},       // 2x
        {31,54,34,36,36,37,38,40,42,45,48,51,56},       // 3x
        {33,34,57,38,38,38,40,41,43,46,49,52,56},       // 4x
        {34,36,38,60,40,41,42,43,44,47,50,53,58},       // 5x
        {34,36,38,40,63,43,44,45,46,48,51,54,58},       // 6x
        {35,37,38,41,43,66,46,47,48,50,52,55,59},       // 7x
        {37,38,40,42,44,46,69,48,50,52,54,56,60},       // 8x
        {39,40,41,43,45,47,48,72,52,53,56,58,61},       // 9x
        {42,42,43,44,46,48,50,52,75,55,57,60,63},       // 10x
        {44,45,46,47,48,50,52,53,55,78,58,61,64},       // jx
        {47,48,49,50,51,52,54,56,57,58,80,61,64},       // qx
        {50,51,52,53,54,55,56,58,60,61,61,82,62},       // kx
        {55,56,56,58,58,59,60,61,63,64,64,65,85}        // ax
    };


    public static double value(card.Rank one, card.Rank two, boolean suit){
            if(suit && one.ordinal() < 2)   return ((not_suited[one.ordinal()][two.ordinal()]+4)    / 100.0);
            if(suit)                        return ((not_suited[one.ordinal()][two.ordinal()]+3)    / 100.0);
                                            return ((not_suited[one.ordinal()][two.ordinal()])      / 100.0);
    }
}
```

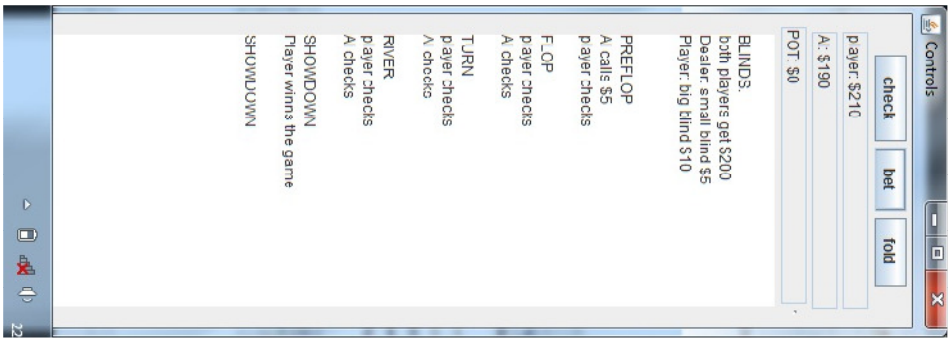
A simple opponent model

```
public class simple_opponent_model {

	public static double[] action(double[] i, String a){
		double[] t = new double[9];

		if(a == "bet" || a == "raise"){
			double o=(i[5]*.1)+(i[6]*.11)+(i[7]*.13)+(i[8]*.15);
			t[5]=i[5]*.9;t[6]=i[6]*.89;t[7]= i[7]*.87;t[8]=i[8]*.85;
			t[0]=i[0]+o*.3;t[1]=i[1]+o*.2;t[2]=i[2]+o*.2;t[3]=i[3]+o*.2;t[4] = i[4]+o*.1;
		}

		else if(a == "check" || a == "call") {
			double o=(i[0]*.1)+(i[1]*0.11)+(i[2]*0.13)+(i[3]*.15);
			t[0]=i[0]*.90;t[1]=i[1]*.89;t[2]=i[2]*.87;t[3]=i[3]*.85;
			t[4]=i[4]+o*0.3;t[5]=i[5]+o*.2;t[6]=i[6]+o*.2;t[7]=i[7]+o*.2;t[8]=i[8]+o*.1;
		}

		else	return i;
			return t;
	}

}
```

Images

Chapter 2

1: the controller of the GUI (turned 90 degrees)



2: the board of the GUI

Literatuur

References

[1] Approximating Game-Theoretic Optimal Strategies for Full-scale Poker - D. Billings, N. Burch, A. Davidson, R. Holte, J. Schaeffer, T. Schauenberg, and D. Szafron - Department of Computing Science, University of Alberta.

[2] Computer poker: A review, Jonathan Rubin & Ian Watson, 2010

[3] SoarBot: A Rule-Based System For Playing Poker, Robert I. Follek, Pace University

[4] Opponent Modeling in Poker, Karen Glocer & Mark Deckert, June 15, 2007.

[5] A survey on artificial intelligence in stochastic games of imperfect information: poker - Deniz Dizman

[6] What is Bayesian statistics and why everything else is wrong, Michael Lavine, ISDS, Duke University, Durham, North Carolina.

[7] Opponent Modelling in Heads-Up Poker, Timm Meyer and Dr. Jonathan L. Shapiro. Philipps-University of Marburg, Department of Mathematics and Computer, University of Manchester, School of Computer Science.

[8] Opponent Modeling, Ryan Carr, February 2009 / CMSC 634

[9] Measuring the Performance Potential of Chess Programs, Hans J. Berliner, Gordon Goetsch and

Murray S. Campbell.

[10] Hold'em Brain – King Yao – 2004

[11] Mathematics and Poker - Brian Alspach

[12] Using Probabilistic Knowledge and Simulation to Play Poker - Darse Billings, Lourdes Peña, Jonathan Schaeffer, Duane Szafron - Department of Computing Science, University of Alberta

[13] Hold'em Poker For Advanced Players - David Skalinsky and Mason Malmuth

[14] Dealing with imperfect information in poker – Denis Richard Papp.

[16] Game theory and AI: a unified approach to poker

Games - Frans Oliehoek

[17] The challenge of poker - Darse Billings, Aaron Davidson, Jonathan Schaeffer ∗, Duane Szafron.

[18] Pokerrobots - AI met een pokerface - Alexander Franciscus Wattimena - Augustus 2006

Literatuur

Sites

[1] http://yudkowsky.net/rational/bayes

[2] http://www.yourpokerbot.nl/wat-is-een-pokerbot.html

[3] http://pokerai.org

[4] http://www.ai.rug.nl/~mwiering/Tom_van_der_Kleij_Thesis.pdf

[5] http://www.cbloom.com/poker/

[6] http://www.csse.monash.edu.au/bai/poker/

[7] http://www.gamecareerguide.com/features/896/poker_ai_a_starting_.php

[8] Learning to Play Strong Poker, Jonathan Schaeffer, Darse Billings, Lourdes Peña, Duane Szafron, University of Alberta

Pokerbots

[1] GoldBullion

[2] Bayesian poker player

[3] poki

[4] loki