Imperial College of Science, Technology and Medicine
(University of London) Department of Computing

A No Limit Texas Hold'em Poker Playing Agent

Lawrence R. Booker
lrb03@doc.ic.ac.uk

Project supervisor : Dr. William J. Knottenbelt
Second marker : Professor Keith L. Clark

September 2004

Submitted in partial fulfilment of the requirements for the MSc Degree in Computing Science of the University of London and for the Diploma of Imperial College of Science, Technology and Medicine.

ACKNOWLEDGEMENTS

I would first of all like to thank my supervisor, Dr William J Knottenbelt, for suggesting this project, and for his continued enthusiasm for playing and analysing the various implementations contained herein.

I would like to thank my family for supporting me, particularly by providing me a place to stay and being there for me when I needed it.

To my fellow lab rats – Claudio, Dan, Douglas, Floric, J P, Luke, Nic, Riccardo, Saar – I will miss the 9 am to 11 pm grind with you guys every day. I don't think that I've ever worked so hard. To all my other friends from the course – you know who you are – thank you for being there for me.

Finally, I would like to thank Steve Spencer and Claire Furney for the exchange of ideas and jokes, it was fun working on the joint sections of the project together.

ABSTRACT

Poker provides an excellent benchmark to study and evaluate cognitive models in tractable yet naturalistic settings that are simple and formal yet reproduce much of the complexity of real life. It is probably the most widely played card game, with endless variations played by millions of adherents from casual players gambling pennies to professionals competing in million-dollar tournaments. Unlike other games that emphasize one particular aspect of cognition, poker involves a broad range of cognitive activities, including:

1.Reasoning under uncertainty (opponents' cards)
2.Dealing with probabilistic outcomes (future cards)
3.Decision-making with multiple options (chips used for bets)
4.Individual differences (different styles of play)
5.Inference of intent (from opponents' bets)
6.Intentional deception (bluffing, sandbagging)
7.Pattern recognition (detecting trends from flow of game)
8.Economic behaviour (factoring impact of amount of bets)

Because of the range of cognitive activities involved, poker provides a broader and more challenging test for cognitive modelling than other games such as chess that focus on a more restricted range of mechanisms (e.g. search). Despite the complexity of aspects involved, it remains a highly tractable domain, partly because it abstracts away from computationally demanding perception and interaction problems. Poker is increasingly being played in online gaming communities where the need for challenging, cognitively plausible agents is increasing. Poker therefore provides a challenging domain at the intersection of fundamental research questions and potential mass application.

To this end, it is desirable to build a no limit poker agent, playing Texas Hold’em, which displays evidence of these cognitive activities. This project attempts to build such a poker bot, with the hopes of competing to the highest level amongst known No Limit Poker bot players, and to play a decent game of poker against a human.

TABLE OF CONTENTS

Section 1 : Introduction

1.1 AIMS OF THE PROJECT

The aim of this project is to build a Poker bot that satisfactorily demonstrates most or all of the cognitive activities mentioned in the abstract within the game of Texas Holdem No Limit Poker. Any bot that meets these aims should be able to play a good game of Poker against both other bots, and, more importantly, humans. Analysis of bots created with different aspects of the above in mind should give an idea of refinements that might improve the play of each implementation. The best-implemented bot could also be put up for possible entry into the ICCM PokerBot World Series [1].

This report starts with the background research, describing firstly the rules and ideas behind the game of poker. The research then turns to the idea of human strategies within the game of poker, and finally to machine learning, and the various approaches to building a poker agent.

Then, the report concentrates on the design and implementation of the server and client, before progressing onto the design and implementation of each of the bots. The results of the tests run on each bot are given, with analysis of the results presented in each case.

The results of a multiplayer tournament completed against the poker bots built by others who have attempted the same project are presented, along with various tournaments against a human player. Finally, a discussion of the project as a whole, along with any conclusions to be drawn and suggested future extensions are presented.

Section 2 : Background of Poker

Before attempting to explain any implementations of bots or protocols, firstly the rules of Texas Holdem must be explained.

2.1 RULES [2]

The Shuffle, The Deal and The Blinds

The dealer shuffles a standard 52-card deck.
(In casinos, the dealer never plays. A round disc -- known as a "dealer button" -- moves clockwise from player to player with each hand. The button marks which player would be the dealer if the deal were advanced from player to player as the game went along.)
Games start with the two players to the left of the dealer (the button) putting a predetermined amount of money into the pot before any cards are dealt, ensuring that there's something to play for on every hand. This is called "posting the blinds." Most often, the "first blind" -- the player to the left of the dealer -- puts up half the minimum bet, and the "second blind" puts up the full minimum bet.
Each player is dealt two cards; face down and unseen by the other players. These are known as their hole cards.

Betting Begins

A round of betting takes place, beginning with the player to the left of the two who posted the blinds. Players can call, raise, or fold when it's their turn to bet. A call (known as a check when the amount to call is zero) means that the player will match the highest bet so far put into the pot, minus whatever they have already contributed. A raise means that the player will call whatever bet has already been made, and raise the bet by a further amount. A fold means that the player forfeits all money in the pot, throws in their hole cards, and waits for the next game. The round of betting continues until every player has either called or folded the last bet/raise, with the consequence that a player can’t re-raise himself.

The Flop

After the first betting round, the dealer discards the top card of the deck. This is called burning the card and is done to ensure that no one accidentally saw the top card, and to help prevent cheating.
The dealer then flips the next three cards face up on the table. These cards are called the "flop."

NOTE: Eventually, a total of five community cards will be placed face up on the table. Players can use any combination of the community cards and their own two cards to form the best possible five-card Poker hand.

After the flop, another round of betting takes place, beginning with the player to the left of the dealer (the button). During this and all future rounds of betting, players have to choose whether to check, call, raise, or fold when it's their turn to bet.

The Turn

The dealer burns another card and places one more card face up onto the table. This, the fourth community card, is called the "turn" or "Fourth Street."
The player to the left of the dealer (the button) begins the third round of betting.

The River

The dealer burns another card before placing the final face-up card on the table. This card is called the "river" or "Fifth Street."

Final Betting and The Winner

Players can now use any combination of seven cards -- the five community cards and the two cards known only to them -- to form the best possible five-card Poker hand.
The fourth and final round of betting starts with the player to the left of the dealer (the button).
After the final betting round, all players who remain in the game can reveal their hands. The player who made the last raise in the river round shows their hand first. A player can choose not to show their hand if they can’t beat a hand already shown. The player with the best-ranked hand wins.

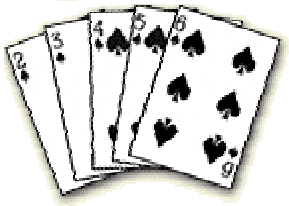
Split pots in betting

It is possible for a player to run out of money whilst betting, rendering him unable to continue betting even though other players might wish to keep betting. In the event that this happens, the concept of split pots is introduced. The main pot consists of all money so far contributed by the other players to the pot up to the amount that the player has so far contributed, plus any other money that was bet by people that have already folded. This is under contention between all players. Then, a side pot is created for all the people who still wish to bet. This pot is only filled by money from those players that still wish to bet and have enough money to do so, and is therefore also only under contention between those players that contributed to it. If the situation arises again, a further side pot is created, and so on.

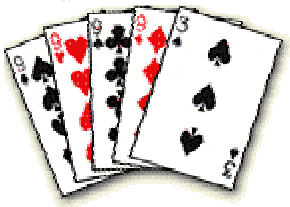
2.2 HAND RANKS [3]

Standard five-card poker hands are ranked here in order of strength, from the strongest hand to the weakest.

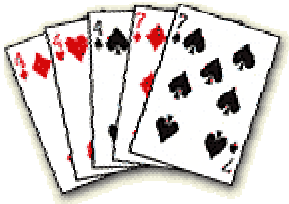
Royal Flush:
Ten, Jack, Queen, King, Ace of the same suit.



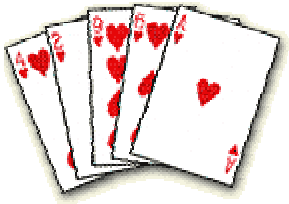
Straight Flush:
Straight with all five cards in the same suit.



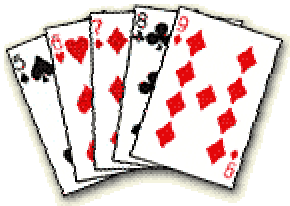
Four of a Kind:
Four cards of the same number or face value ("quads").



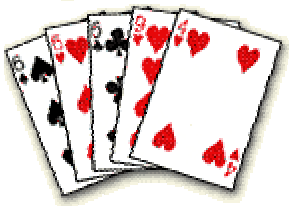
Full House:
Three cards of one number or face value and two cards of another number or face value. If more than one player has a full house, the full house with the highest ranking three of a kind ("trips") wins.



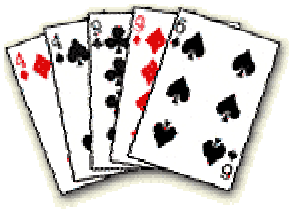
Flush:
Five cards of the same suit. If there is more than one flush, the hand with the highest card(s) wins.



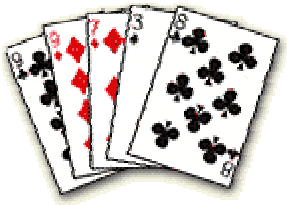
Straight:
Five cards in sequence. Cards can be in any suit. An Ace can be used in the highest straight (10, J, Q, K, A) and the lowest straight (A, 2, 3, 4, 5).



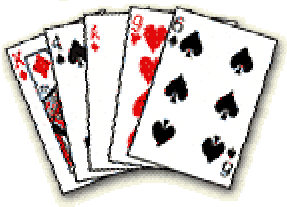
Three of a Kind:
Three cards of the same number or face value ("trips").



Two Pair:
If two players have two pair, the hand with the highest pair wins. If they have the same high pair, whoever has the second highest pair wins. If they have the same two pair, whoever has the highest fifth card ("kicker") wins.



One Pair:
Two cards of the same number or face value. If two players have the same pair, the highest outside card(s) wins.



High Card:
The hand with the highest card(s) wins.

Figure 2.0.1

How is the best hand rank determined?

Between two hands, the hand with the highest rank (above) is the best. In the event that both hands have the same rank, the hand with the highest kicker is determined to be the best. If both hands contain the same valued kicker, the next highest card is used for comparison. This continues for all five cards in the hand. If neither hand is better after these tests, they are considered to have an equal rank, and will split the pot that they are involved in.

2.3 TERMINOLOGY [4]

- Blinds - Texas Holdem Poker uses what's called a "blind" structure, meaning that two people on the table must post a bet prior to seeing their cards. Since they are forced to bet without seeing their cards, they are playing "blind", thus the name of those bets are called "blinds". There are two blinds, the big blind and the small blind. The small blind position must post half the minimum bet and sits immediately to the left of the dealer. The big blind must post the full minimum bet, and sits immediately to the left of the small blind, two seats to the left of the dealer. As the deal rotates around the table, each player takes turns posting the small blind and the big blind bets. This blind structure forces the action on the table since there will always be a pot to win. So, for example, if you are seated at a £1-2 limit Holdem table, the small blind must post £.50 and the big blind must post £1 bet. As play rotates around the table, each player may choose to call that £1 bet, raise, or fold. When it's the small blind's turn, that player only needs to call £.50 to play the hand.

- Board - The board refers to the community cards that are dealt face up on the table. In Texas Holdem, there will ultimately be five community cards on the "board". The board does NOT include the two private cards dealt to each player. So, if someone were to say, "the board plays", the player means that the five community cards make his best poker hand and he is not using any of the two private cards dealt to him.

- Button - Also called the "Dealer Button", this is a white disc that signifies the dealer's position on the table. The dealer's position is significant because he is the last player to act for that hand. The Dealer Button rotates around the table, so each player takes turns being "on the button".

- Check-raise - the act of checking a hand, in hopes of luring your opponent to bet, so that you may then raise over him and build a bigger pot to win.

- Flop - In Texas Holdem, each player has two cards dealt to them, and then share five community cards. These five community cards, however, do not all get dealt at the same time. There are rounds of betting at certain intervals during the deal. After the first two cards are dealt to each player, there is a round of betting. Then, three of the five community cards are dealt at one time on the board. This is what's known as the "flop" - the first three cards being dealt on the board. The fourth card is called the "turn", and the final, fifth card is known as the "river".

- Flush Draw - a hand where you have 4 of the 5 cards needed to make a flush. For example, if you are holding two clubs, and the board flops two more clubs, you would be holding a flush draw. You would need to draw an additional club to complete the flush.

- Inside Straight Draw - a hand where you have 4 of the 5 cards needed to make a straight, but your 4 cards are not "connected" or in sequential order, so you need a single card in the middle of your straight to complete the straight. i.e. - you are holding 5 6, and the board shows 7, 9, 10. At that moment only an 8 will make your straight.

- Kicker - the kicker refers to your tie-breaking card. For example, if I am holding an Ace and King, and the board shows Ace, 5, 8, 2, 6 - I would have a pair of Aces with a King kicker. My opponent may have an Ace also, but with a weaker kicker, in which case I will win the showdown.

- Implied Odds - Implied Odds refers to the odds or percentage of making your winning hand compared to the odds or percentage of the bet you must call compared to the POTENTIAL size of the pot.

- Limp - to limp into the hand refers to calling the minimal bet, the big blind, to play your hand.

- Loose player – A loose player is a player who takes a lot of hands to the flop

- Maniac - A maniac is a player who plays ultra-aggressively, making big bets on poor cards, throwing his money around in the hopes of winning by bluffing and intimidation. A maniac can be a difficult player to play against, but may also be a big source of profits.

- Muck - the act of folding one's hand without showing your cards.

- Nuts or Nut Hand - the Nuts or Nut hand is the best possible hand at that particular moment. For example, if I am holding the Ace and Jack of hearts, and the flop brings the 2, 6, 9 of hearts, I would be holding the "nut" hand or just the "nuts" - there is no hand that I can lose to at this moment. Of course, the nut hand can change as the

fourth and fifth cards come out (a straight flush). The highest possible straight or flush is often called the "nut" straight or "nut" flush.

- Outs - The number of "outs" refers to the number of cards in the deck to make your winning hand. For example, if I have a flush draw (4 suited cards), I have 9 outs to make my flush since there are a total of 13 cards of each suit.

- Pot Odds - Pot Odds refers to the odds or percentage of making your winning hand compared to the odds or percentage of the bet you must call compared to the CURRENT size of the pot.

- River - The "river" is the dealing of the fifth and final card of the five community cards dealt in Texas Holdem. The "river" is also known as "Fifth Street". The river card is the fifth and final card on the board. See "Flop".

- Semi-bluff - The act of betting on your hand when your hand is not made yet. ie - you have four cards to a straight or flush and you make a bet or raise a bet - even though you do not have a strong hand now, you have a chance of bluffing your opponent out of the pot or hitting the card that will complete your winning hand.

- Short-handed - A short-handed game refers to a table that has few opponents. A table of 2-6 players (as opposed to a full table of 10) would be considered a short-handed game.

- Slow-play - The act of intentionally under-playing a very strong hand in the hopes of tricking your opponent into thinking that he has you beat, which leads to your opponent betting more in later rounds. i.e. - holding pocket Aces and just checking or calling on the first round of betting to show weakness, in hopes of luring your opponent into more aggressive play later and a larger pot.

- Smooth Call - A smooth call is the act of just calling a bet or raise with a very powerful hand in order to conceal your strength and keep all the attention on the initial raiser.

- Stone Cold Bluff - This is the act of betting your hand with no real possibility of winning the hand if your bet is called. Unlike the semi-bluff, where you still have the potential to make a winning a hand, the stone cold bluff is not relying on the cards, but on your opponent folding.

- Suited Connectors - Holding two cards that are of the same suit and sequentially ordered. i.e. - 8, 9 of spades.

- Tight Player – A tight player is a player that is very selective about the hands that they take to the flop

- Turn - The "turn" is the dealing of the fourth card of five community cards dealt in the game of Texas Holdem. The "turn" is also referred to as "Fourth Street". The turn card would be the fourth card on the board. See "Flop".

Section 3 : Strategies

3.1 NO LIMIT VS LIMIT POKER

Limit Poker

Limit Texas holdem "limits" the amount that can be bet in each of the four betting rounds. When playing at a £2-4 limit table, the small blind will be £1 and the big blind will be £2. The limit for a bet or raise is £2 pre-flop, and £2 after the flop. On the turn and river, it is only possible to bet or raise £4. So, all the betting increments on the first two rounds of betting are consistently £2, and all betting increments on the third and fourth rounds are consistently £4. Each round is limited to three raises in total.

No Limit Poker

”No limit” Texas holdem has no limit to the amount allowed to be bet. At a £0.25-0.50 no limit table, the small blind is £0.25, the big blind is £0.50, the minimum bet for all four rounds of betting is £0.50, and the maximum bet for all rounds is whatever a player has in their bankroll. At any point in a hand, a player has the ability to bet their entire bankroll by going all-in at any time. There are also no limits to the number of raises.

The game of limit Texas holdem is much different than the game of no limit holdem. Although the rules are the same, the dynamics drastically change.

3.2 IMPORTANT GAME DYNAMIC DIFFERENCES BETWEEN NO LIMIT AND LIMIT POKER A PLAYER MUST RECOGNISE

Limit Poker

When playing in a limit game of poker, bets are restricted, either to the size of the pot, or to a particular multiple of the blind. This allows for players to be fairly loose, as they won’t be punished too heavily for attempting to hit an unlikely hand on the flop with weak hole cards. Limit also means that a player can call to showdown with weaker hands, as again the penalty is relatively mild. These two observations lead to the amount of overall bluffing being reduced to a lower level, since the bluffs are more likely to be called out and exposed.
How limit poker affects the building of a limit poker agent

This reduction of risks and a more “honest” game allows for the making of a bot to be fairly trivial. All that has to be done is to create a bot that understands how its hand ranks on average compared to the average expected hand for the game. If the hand is very good compared to the expected hand, it should raise, if it is average, it should call, and if it is poor, it should fold. In general this strategy can lead to a reasonable limit bot.

No Limit Poker

When playing a no limit poker game, however, the situation is very different. Players can bet their entire bankroll preflop, meaning that it can become very expensive to even see a flop. Limpers can find life very hard-going, and aggressive players have a much higher chance of pushing players off of the pot than in limit poker. Bluffs are less likely to be called, since a high bet can actually be real, and the unfortunate who calls can be caught short for their entire bankroll and knocked out of the tournament.

How limit poker affects the building of a no limit poker agent

A bot that plays reasonable limit poker can find itself calling an all-in with a reasonable hand and losing the its bankroll every time. A corresponding raise in the probability above which the bot will play leaves it losing when the blinds raise later in the tournament, and stranded with no money waiting for a decent hand to come along.

The recognition in the limit bot of how the pot affects whether it will bet or not can also be exploited. A huge raise will normally lead to decent hands being folded by a limit bot, unless the hand has a very high probability of winning.

Savvy opponents also easily spot a rock player, which is essentially what the limit bot tends to be. All that the opponents have to do is fold when the bot raises, and force it off the pot at any other time by bluffing and raising heavily.

These difficulties put opponent information at a premium. A bot will need to quickly and accurately determine what type of player each opponent is, and will also need to accurately guess what type of hand each opponent has based on their previous actions and the current game situation.

NOTE ON BACKGROUND RESEARCH

It was decided that because of the outlined differences in dynamics above, instead of concentrating research into the area of previous implementations of limit bots, a fresh approach was needed. In this respect, research was concentrated into human strategies within no-limit poker, and how to introduce these ideas into any implementation.

3.3 HUMAN STRATEGIES IN NO LIMIT POKER

GENERAL IDEA IN BETTING

The main idea behind poker is to maximise the amount of money that you can win from the table given any hand. Players will want to make sure that they go to the showdown with their strongest hands whilst getting as much money put into the pot from other players as possible. On the other hand, a player who is bluffing will want to make sure that he doesn't have to show his hand – i.e. he will want to try to force players with better hands off of the pot by betting big and making his hand seem better that it actually is. In general, a player who has the nuts at any particular point in the game will want to try and win the pot there and then if his hand has a chance of decreasing in strength later, and a player with a chance of hitting a big hand later will

want to try and stay in the game for as little money as possible, then try to bet big when they know that they have the nuts or a high chance of having the best hand.

POINTS OF CONSIDERATION FOR NEW PLAYERS [5]

TECHNICAL SKILLS

There are basic technical skills that provide a framework for decision-making under pressure, and are essential, especially for beginners.

STARTING HAND SELECTION

Starting hand charts are useful for beginners, but more experienced players might play lower ranked hands in certain situations. For instance knowing whether hole cards 8 9 suited is a playable hand in a certain situation, or whether it isn't (hand charts would say that this isn’t). Experience can say which hands play better with just a few callers, and which ones play best with more callers.

Deciding which starting hands to play is adjustable for the situation. Taking 18% of hands to the flop may be appropriate for a certain situation, or could be too loose, particularly in an aggressive game. Playing 35% of hands might be too tight if the game is passive. Selecting which starting hands to play relies on knowledge of when, why and how to get started.

PROBABILITY AND ODDS

Basic statistics play a role in almost every decision, so knowledge of statistical analysis and probability is important. Calculating probability based on favourable outcomes (outs) is a start. Knowing the odds of hitting on the flop is also important (but difficult to do in your head).

The ability to calculate expected value, that is, the amount that you can expect from a particular pot when making a certain sized bet, is complex enough that a good understanding of when it applies and how to calculate it is valuable.

READING HANDS

This is the ability to quickly recognize the likely hands that an opponent may hold. At its most basic level it requires knowledge of what is possible. New hold 'em players often bet two pair or three of a kind against a board that suggests a flush or straight, whereas an experienced player would recognise the danger and might not.

At a deeper level, the ability to accurately recall the order of play: who bet, who raised, who re-raised - not just on this betting round, but during the entire hand, provides additional information in determining what is possible. This is almost always just partial information, and depends on several factors like the quality of the competition and how extreme the action has been. A good player is unlikely to hold a full house when the flop is 4♥ 4♠ 7♦ (i.e. the player is unlikely to hold a 4 7), unless they were on the blind and weren’t raised, whereas a newer player might. If there was heavy raising before this flop, it is less likely that another player has a full house.

Technical skills can be learned through self-study, reading and practice. Anyone with an average aptitude for maths and card playing experience can master them with a little bit of effort. Most above average players have a full understanding of the technical aspects of the game.

SOFT SKILLS

GAUGING THE PLAYER

This is the ability to quickly and accurately determine how much poker a person has played, how good he is, and what his playing style is. Further, the ability to validate or adjust these assumptions based on further actions by this player is important. This is an evolving situation in which one can make some initial assumptions, and then gain confidence in them as one continues to learn about the player. As evidence to support these assumptions grows, so does one's ability to take actions based on it.

Is the player:

- Aggressive or conservative?
- Loose or tight?
- A pot stealer – i.e. someone who often bets from a late position when no other bets have been made?
- A trapper or slowplayer?
- A loose caller?
- One who always bets draws?
- One who bets appropriately for the strength of his hand?
- One who varies his play, i.e. plays tight when the table is aggressive, but loosens up when appropriate?

Quickly determining what type of player an opponent is adds a dimension to reading hands that can be used to take actions that may otherwise be questionable, even if it's only a rough estimate. The size of the risk taken should be appropriate for the amount of confidence you have in this estimate. For example, you might make a call equal to twice the big blind with initial assumptions, but decline to make a call equal to ten times the big blind until you are more confident in what you know about that player.

READING THE PLAYER

The ability to read a player is the ability to predict a player's hand, based on the kind of player that you think he is along with his actions, and factor that into your play. This means having a feel for what a certain action means for a specific player. For example, an all-in bet from some players is a blatant attempt to force their opponent

out of the pot. When these players want to build the pot because they have a strong hand, they don't force others out by going all-in. For other players, an all-in bet may mean that they have a strong hand, and have put their opponent on a good second best hand, and they want the call.

Alternatively, a soft bet on the flop may be a trap, or a weak attempt to steal when no one is likely to have hit. Most average players are relatively consistent in how they play their cards, and once you have gauged their play it can be a strong factor in how you react to their actions. Even above-average players can be predictable to a degree.

PLAYING THE PLAYER

This takes reading the player to the next level. It is the ability to use your knowledge to get a specific action from a certain player that you are familiar with.

For example:

Some players hate to be re-raised and have to dominate the hand. Against such a player, if you have a strong hand and want to isolate him, a possible tactic is to re-raise him pre-flop. Often he'll go over the top, maybe all-in. To avoid guesses at this strategy, the next time he bets, put out a small raise with anything, and then fold to his re-raise.

Against a “pot stealer” when you have a strong hand, a possible tactic could be to check the flop and weak bet on the turn to induce a steal or a bluff, then call his raise, and bet big or go all-in on the river. “Pot stealers” hate to think that anyone else would use this tactic against them, and they might call to the finish or come back even stronger.

DECEPTION

Deception plays a very important part in poker strategy: many pots can be won with a hand that is not the best at the table, and the value of a pot can also be increased with the correct play. There are several strategies involved in deception:

Bluffing

This is the act of representing a much stronger hand than you actually hold, with the aim of scaring an opponent off of a pot, or deceiving your opponent into thinking that your style of play is different to what it actually is.

A semi bluff is used in order to raise a pot with the hope of hitting a hand that isn’t currently held, with the hope that the hand will come up with future board cards.

A stone cold bluff is used even when there is no chance of hitting a good hand. This is used purely to try and scare opponents off the pot.

Slowplay / Sandbagging

This is the act of delaying big betting until later rounds, even though it is known that the current hand is good at an earlier stage. For instance, if an ace is held preflop, and a pair of aces come down on the flop, if a player were to call on the flop, and then raise on the turn or the river, this would be sandbagging, as the player did not immediately react to the fact that they had a good hand. This strategy can lead to more players staying in the pot until a later round, possibly deceiving them as to the real strength of the hand, and increasing the eventual amount of money to be won in the pot.

Check/Raise (a type of sandbagging)

This is the act of checking in a round, and then when another player has made a bet, raising within the same round. This puts the other players in a difficult position, as they have already committed money to the pot, and could be reluctant to bet in a round and not see the next board card. Again, this is made to increase the size of the pot.

CHARACTERISTICS OF A PLAYER

As mentioned above, essential to a players' poker strategy is the ability to gauge and analyse the other players at the table. Of course, there is no way of knowing at any time what cards your opponents actually hold, but there are many characteristics of their play that can give some kind of indication as to what type of player they are.

Tight/Loose Player

Keeping track of where another player folds can tell you how likely he is to have a good hand when he plays. A player who folds a high percentage of the time before the flop is deemed to be a tight player, one that is more likely to have a good hand when he actually does play a hand. A player who sees a large percentage of flops is deemed to be a loose player, that is he will be less likely than a tight player to have a good hand when he plays a hand.

Conservative/Aggressive Player

Keeping track of how a player bets post flop can give an indication of whether he is an aggressive bettor or a conservative bettor. This can normally be determined by how often a player bets post flop, and also taking note of what hole cards they had if it comes to a showdown. A consistently aggressive bettor is more likely to be bluffing than a conservative bettor when they make a large bet.

SPECIFICS OF BETTING STRATEGY

Human strategies in No-Limit Poker can normally broken down into how a player plays at each stage of the game. Normally a player will have a set of specific things to look for on the preflop, a different set of things to look for on the flop and so on.

Pre-flop

A common strategy for poker players on the preflop is the use of Sklansky's analysis, that is, the 169 possible hole card combinations are placed into groups of descending strength, with Group 1 containing the most powerful combinations, all the way to Group 8 with the least powerful.

[6]

Here is a list of the generally accepted groups, where an s by the cards denotes that they are of the same suit, and no s denotes that they are of different suits:

Group 1: AA, KK, QQ, JJ, AKs

Group 2: TT, AQs, AJs, KQs, AK

Group 3: 99, JTs, QJs, KJs, ATs, AQ

Group 4: T9s, KQ, 88, QTs, 98s, J9s, AJ, KTs

Group 5: 77, 87s, Q9s, T8s, KJ, QJ, JT, 76s, 97s, Axs, 65s

Group 6: 66, AT, 55, 86s, KT, QT, 54s, K9s, J8s, 75s

Group 7: 44, J9, 64s, T9, 53s, 33, 98, 43s, 22, Kxs, T7s, Q8s

Group 8: 87, A9, Q9, 76, 42s, 32s, 96s, 85s, J8, J7s, 65, 54, 74s, K9, T8

(Sklansky 1976)

Normally a player will have these groups in mind when they receive their hole cards, and, together with information on how many opponents, the amount in the pot, the amount to call etc; will perform an action based on which group their hole cards fall into.

This table, however, was formulated with limit poker in mind. There are some subtle changes that have been picked up by various other people [7] [8] whilst playing no-limit poker, which has led to these groups being changed slightly, and in some cases split to recognise further the strength of some hands compared to others.

Full revisions of these are detailed below.

Group 0: AA KK

Group 1: QQ JJ AKs

Group 2: TT AK AQs AJs KQs

Group 3: AQ 99 ATs KJs QJs KTs

Group 4: 88 AJ KQ QTs A9s JTs AT A8s

Group 5: KJ 77 QJ KT QT JT A7s K9s Q9s T9s J9s

Group 6: 66 55 44 33 22 A5s A6s A4s A3s A2s

Post Flop

After the flop, any strategy will depend on many factors, including information gathered pre-flop, what is known about the opponents and how they play, and the probabilities of having or making the best hand. If one has hit top pair with a strong kicker on the flop, then this is a strong hand and the pot should be bet at. But a more difficult scenario is when one has a flush draw or a straight draw on the flop, or perhaps an inside straight draw with two over cards. If someone bets, should the call be made?

At this point, it is important to first surmise what the opponent's hand is. Then, the number of favourable outcomes (outs) that make the hand a winning hand must be calculated. The final step is to then understand the probabilities of hitting one of the outs, giving the winning hand. These are all crucial steps in the decision making process.

Number of Outs	**Percentage of Hitting on either Turn or River**
1	4.4
2	8.4
3	12.5
4 (Inside straight draw)	16.5
5	20.3
6 (Two overs)	24.1
7	27.8
8 (Open ended straight draw)	31.5
9 (Flush draw)	35.0
10	38.4
11	41.7
12 (Flush draw + Gut shot)	45.0
13	48.1
14	51.2
15 (Straight Flush draw)	54.1
16	57.0
17	59.8
18	62.4

Number of Outs	Percentage chance of Hitting on River
1	2.2
2	4.3
3	6.5
4 (Inside straight draw)	8.7
5	10.9
6 (Two overs)	13.0
7	15.2
8 (Open ended straight draw)	17.4
9 (Flush draw)	19.6
10	21.7
11	23.9
12 (Flush draw + Gut shot)	26.1
13	28.3
14	30.4
15 (Straight Flush draw)	32.6
16	34.8
17	37.0
18	39.1

For example, if one does not hit the desired card on the turn, can it be assumed that Player1 will bet again, and if they do, how much will they bet? If this information is known, this should also be included in calculations of the bet vs. pot amounts.

Suppose that it's known Player 1 will probably bet another £2 after the turn. So essentially, calling £4.00 is weighed against a pot of £12.00. The percentage now is 33% (4/12) as opposed to the earlier example where it was 20%. In this case, the call isn't as clear as before and the current pot odds don't quite justify the call because 33% is greater than our 31.5%. However, there are implied odds to consider, and these odds may justify making this call.

This is the last factor that is important to consider – how big will the pot be by the river? In the game of no-limit holdem, the pot could eventually be much larger than the current pot against which the odds are calculated. If the desired hand is hit, if the entire pot amount was bet, could it be assumed that Player 1 will call? How much of a bankroll does Player 1 have and is it possible to take it all on this hand? These are interesting questions and also can affect the decision of what action to take. The total pot size at the very end of the hand could easily justify making the call in the hopes of winning that pot. This is called "implied odds", and should also be considered.

3.4 AUTOMATED STRATEGIES IN NO LIMIT POKER

GENERAL IDEA IN BETTING

From a strictly mathematical (i.e. a bot's) view, the aim is to maximise the expected value of the pot. This depends on how players at the table react to certain sized bets in particular situations. There are proposed formulae

[9]

for the appropriate action to take in these situations, in order that the pot size can be maximized. These, however, are based on the betting strategies appropriate for limit poker, and hence hold little or no consideration for the dynamics of no limit betting. For this reason, initially, a bet that's proportional to the perceived strength of a hand can be used. This can later be capped to better mimic the slowplay betting of a human player.

STARTING HAND SELECTION

From the point of view of a rule-based bot, a hand chart containing all possible hole cards is an obvious basis for betting strategy. As discussed for a human player, the Sklansky groups (or a refinement of the Sklansky groups) can be used, and actions based on which group the hole cards are in can be performed. This strategy can be extended to take into account hole cards that other players have played with, and attempt to loosen up or tighten up on the appropriate tables.

A bot has an advantage, however, of being able to run simulations of potential situations in order to estimate the chance of its hole cards winning. Techniques to accomplish this are described in more detail in the section below.

Either starting hand charts or a simulation could be used in order to decide which starting hands are played.

PROBABILITY AND ODDS

In this area, where a new human player might struggle, a computer-controlled bot can excel. As long as the formulae provided are correct, and the techniques sound, a bot can accurately and quickly obtain appropriate information on odds of its hand winning, the pot odds, hand distributions, and so on. A probability "roll-out" can be performed, pitting the potential hands that the bot could make, given the cards that it has, against any number of other possible hands that an opponent could hold. The higher the amount of hands tested, the more accurate the worked out probability will be. This probability can then easily be used against the amount to call and the pot size in order to work out the pot odds, and see whether a bet is worth making.

READING HANDS

In order to check what hands are possible, an exhaustive search can be performed based on what board cards have (or haven't) come down. The main problem is how to

use the information available. Relying on information about the best possible hand that could come up doesn't benefit bot play much, since the best possible hand is always at worst "4 of a kind" after both the flop and the turn, and this remains an inaccurate indicator for opponents' possible hands.

Better is an overview of all possible hands:

One way is taking an average hand rank for a given set of board cards. The average hand rank can again be worked out using a rollout and possible hands available, and the overview of possible hands can be represented by the variance of the ranks of the hand ranks from the mean. This technique does not evaluate every possible hand, but a random sample large enough to accurately describe the distribution.

A second way could be to do a rollout simulation, giving the probability of a hand winning at any given stage, which can also give an overview of the situation for given board cards.

GAUGING THE PLAYER

A human player might attempt to guess at what a player might hold when they perform certain actions, and refine their assumptions as time goes on. A computer bot, with no sense of how to play against particular players and without sharp analytical skills, must rely on statistics. For this reason, the more complete a set of statistics; the better a bot can make a decision. Any hole cards exposed can be logged, along with the actions that the player performed with them, as well as the board cards and the possible hand distribution with these cards. Using these statistics, a bot can attempt to predict the hole cards that the opponents at the table are likely to have, given their actions.

READING THE PLAYER

In order to "read" a player, i.e. to determine whether some specific actions will flag exactly what type of hand an opponent has, a bot would have to be programmed with a specific set of actions to take note of. This is a problem, as these would be guesses, and would apply to specific players at best. In order for a bot to learn about these, it would have to run through the specific actions and call to showdown to check whether these assumptions are correct. This is a very specific type of investigatory play, and for the purposes of this project, less profitable than an attempt at a generally adaptive style of play. A subtle approach is needed, which could be attempting to adapt in the same way that humans who are trying to get better at poker would adapt.

PLAYING THE PLAYER

This involves trying to maximise the pot by attempting to trick a player into betting at an inappropriate time. Again, this is a subtle intricacy of human play that would be very difficult to replicate in the bot player, given the amount of unknowns, and the difficulty in succinctly describing a certain game situation.

DECEPTION

Bluffing

This could easily be achieved in a bot player: the bot simply has to behave as an always raise player at random times.

Sandbagging

This is slightly harder to achieve in an automated way, since an attempt at sandbagging could leave the bot not forcing other players off of the pot when it should, and ending up with another player hitting a better hand on the turn or the river.

Check/Raise

This also could be difficult to implement, since if a bot has good hand, normally the rules dictate that it should bet an amount. If the bot has not such a good hand, then it should not be raising at all.

ALL STATISTICS GATHERED

- A complete account of all actions that a player has done preflop and postflop
- A complete account of all actions that a player has done preflop and postflop, whilst on the blind
- A percentage breakdown of exactly where a player has folded
- A set of all pairs of hole cards that a player has had when shown at a showdown
- The amount of times that a player has gone to a showdown and not shown their cards

It should be noted that an opponent's hole cards are shown very rarely. On the showdown, if a player chooses not to show their cards, they can muck. Normally only if a player has a winning hand do they show their cards. This means that building up information about the hole cards that players had can take some time, and the results are normally skewed towards the higher possible hand ranks.

Section 4: Machine Learning

4.1 OVERVIEW

As mentioned above, statistics need to be collected on opponents in order for a reasonable poker bot to be made. There does, however, need to be a way to collect these statistics, and ways in which to implement strategies based on these actions. Section 3 explained why previous research into limit poker bots only partially aids research into a no limit poker bot, and this section shall outline why most typical machine learning methods would be inappropriate here.

Typical machine learning idea – heavily paraphrased from

[10]

If there is a classification or prediction task that one wishes to program an agent to perform, then it is likely that a machine-learning algorithm will be of use. In this case, one would need to correctly specify a learning problem in terms of the categorisation required, the examples available, and the background information about those examples. Errors in the data would also have to be considered.

Examples

One could automate the process whereby an agent learns from examples. The agent would learn a way of categorising examples into categories, and this could, in turn, be used for prediction tasks. One of the most important specifications of a machine-learning problem is therefore the set of examples that are provided. These will depend on the particular problem: for instance in medical diagnosis applications, the example set will be the patients. In handwriting recognition, the examples may be graphics files derived from scanning in hand-written letters.

Feedback

Often the problem faced by a learning agent is expressed as a concept learning problem, whereby the agent must learn a concept that achieves the correct categorisation. The concept will therefore be a function, which takes an example as input and outputs the correct categorisation of that example.
Usually, to get a machine-learning agent to learn a concept, one would have to supply both positive and negative examples of the concept. Positive examples - often called just positives - are pairs (E, C) where E is an example from the data set and C is the *correct* categorisation of E. Negative examples - often called just negatives - are pairs of (E, X) where E is an example from the data set and X is an *incorrect* categorisation of E.

For instance, if a learning problem was to learn the categorisation of animals into one of four classes: bird, fish, mammal and reptile, then some positives could be supplied, such as:

positive (cat, mammal).
positive(dog, mammal).
positive(trout, fish).
positive(eagle, bird).
positive(crocodile, reptile).

Along with some negatives, such as:

negative(condor, fish).
negative(mouse, bird).
negative(lizard, mammal).
negative(platypus, bird).
negative(human, reptile).

If a machine learning approach can work without negative examples of the concept, then it is said that it can learn from positives only.

Background Information

The background information, or knowledge, associated with a machine learning problem is the set of concepts which one would hope to see utilised in the concept learned to achieve categorisations. The background knowledge describes the examples supplied. For instance, in the "animals" problem, it is possible to learn a way to categorise animals correctly using concepts such as the habitat in which animals live (land, air, water), the number of legs they have, whether they produce eggs, etc.
Also, background information can include relationships between background concepts. These relationships are analogous to the axioms supplied to automated reasoning programs, and they may help a machine learning program to search the space more efficiently.

Errors in the Data

It is worth remembering that in machine learning, one often has to deal with data taken from the real world, and real world data contains errors. Errors come in various forms, including:

1) Incorrect categorisation of examples, such as saying that a platypus is a bird - it does lay eggs - when it is actually a mammal
2) Wrong values for background information about the examples, such as low quality scanning of hand written letters leading to a pixel which was not written on being scanned as black instead of white
3) Missing data, for instance, examples for which we do not know the value of all the background concepts
4) Repeated data for examples, possibly containing inconsistencies.

Certain learning algorithms are fairly robust with respect to errors in the data, and other algorithms perform badly in the presence of noisy data. Hence it is important to assess the amount of error expected to be in the data before choosing the machine learning techniques to try.

4.2 APPLYING MACHINE LEARNING TO POKER

To better explain the difficulty in applying machine learning to a no limit playing poker bot, a comparison between chess and poker has been described:

Imperfect or perfect information?

A game can be classified as either a game with perfect information or a game with imperfect information. The difference between these is the amount of information available to any player about his opponents. With perfect information, all information about each opponent's game state is available, and with imperfect information, not all information is available about some or all opponents' game states. To better explain, consider the games of chess and poker.

In chess, the entire board is visible to both players. This means that all of the opponent's pieces and their positions are known at all times. This is a game with perfect information [11].

In poker, a player only sees his own hand, and knows nothing of any of the opponents' hands. Of course, it is easily deduced that none of the opponents can hold any of the cards currently held in his hand, but there is no knowledge further than that. Any further deductions about opponents' hands are estimates. Poker is a game with imperfect information.

Game state

Depending on whether a game has perfect or imperfect information, an accurate estimation of a game state may or may not be made.

In chess, since all pieces can be seen, an algorithm (known as an evaluation function) can be used that gives each player a rating, with a higher rating denoting the fact that the player is in a stronger position than the opponent.

In poker, the game state is not dependent totally upon the cards dealt to each player and on the board: the types of player playing the hand also affect the state. Poker is an inherently non-Markovian game [12] (i.e. the state of the game is dependent on the actions that any opponents have performed before). Since the opponents are non-deterministic, and the state of the game is inaccessible (that is, other players' hole cards are not observable until the showdown, and sometimes not even then) the game state cannot be succinctly described at any point. This means that techniques usual to games such as chess are invalid in this environment.

Machine learning

As described above, there are four main elements to machine learning: examples, feedback, background information and robustness in the face of errors in the data set.

Examples

In the game of chess, examples are easily and widely given within implementations of chess computers, especially for *gambits*. This is because a particular game situation can and will arise time and time again: in the case of the gambit, every time a new game is started. All information is displayed on the board, and the computer can be sure that the current game is *identical* to one in memory. This makes the giving of examples to a chess computer a worthwhile activity.

In the game of poker however, one cannot be sure that one situation is identical to another at *any* stage of the game. Consider the situation where the observed board cards are 7♠ 7♣ 7♥ 9♣ 2♥, and the hole cards A♠ A♥ are held.

Situation one: an opponent holds 5♥ 3♣; the player's current hand is the best

Situation two: an opponent holds 7♦ 3♣; the player's current hand is not the best

It becomes quite apparent that holding examples of certain game situations is unsuitable for a poker bot, as the above example highlights.

Feedback

In chess, if a move were being considered, before the move were to be made, the player's rating could be analysed before and after: a drop in rating could be reinforced with negative feedback, and an increase could be reinforced with positive feedback. The concept in this case would be to learn a good strategy for playing chess.

In poker, as it is difficult to rate a particular game state, it is difficult to provide feedback for the bot to learn from. Let us consider, for example, the situation described in the *Examples* section. This is where a problem arises:

If the bot were to fold in the first situation, the hole cards of the opponent would not be shown, and it would be unclear as to what feedback to provide the bot.

If the bot were to call in the first situation, the hand would appear to win, but is this a correct action to take? Could the bot have not raised and won more money from the pot?

If the bot were to raise in the first situation, could it have raised by more and won more money? If the opponent did not call the raise, could the bot have raised by less and kept the other player in the pot?

If the bot were to call in the second situation, the hole cards of the opponent would be shown, and it would be natural to penalise the bot. It would be impossible, however,

to distinguish this situation from the first, and so the bot would be discouraged from taking the correct action if situation one were ever to arise.

It becomes clear that opponent modelling would clarify the state slightly, however, since one situation cannot be said to be *identical* to a previous recorded situation, feedback for a poker bot is difficult to implement from one hand to the next.

Another system of feedback could focus on the monetary return from a certain amount of hands played. This has been tried in various other papers [13] ,with limited success, in the implementation of a limit poker playing agent, and with a very limited game set up. This lack of success, coupled with the broader scope of available actions in no limit poker, leads to the discounting of pursuing a feedback loop that encourages or discourages a bot from performing particular actions.

For a poker agent in general, feedback is very difficult to implement. The difficulty arises when rewarding or discouraging a player: what part of the strategy is one rewarding? As each strategy is a complex collection of rules or a non-deterministic selection of an action from a table, some part of the strategy could be correct, whereas another part might be completely incorrect. For this reason, it is very difficult to link good performance back to strategy.

Thus, automatic reinforcement techniques are not appropriate for this environment.

Background information

In a chess computer, it is assumed that an opponent is trying to win, and further that at any stage; the opponent will make the best possible move from any situation

. If an opponent makes a less than optimal move, all the better for the chess computer. In this way, background information about a player type is unnecessary for a chess computer [11].

In poker, background information is essential to both the poker bot and the contained opponent modeller.

Firstly, the bot needs to be supplied with a strategy. This could be a deterministic: in the form of a set of rules dependent upon various game states; or this could be non-deterministic: a random selection from an array of actions or a deterministic strategy with random functions occasionally over-riding the normal strategy.

Secondly, the opponent modeller needs to be supplied with information on how to classify opponents. These classification rules or guidelines could be the recognised perceptions of how players act according to professional players.

As described in previous sections, depending on the actions that a player has made in the past, a player can be labelled with a particular style; ways in which a player has reacted to particular situations or actions previously can also classify a player.

Errors in the data set

A chess computer could feasibly run into errors in a data set, most probably if an example situation were replicated with different suggested moves in each case. As a player is assumed to be the best possible from any situation, any player who makes a less than optimal move will still be assumed to make the best move in the next situation. This assumption, which at many times could be incorrect, will never cause a problem for the chess computer, as it can still deal with any given situation. Incorrect categorisation would never happen, since one position can be determined to be *identical* to another using a very simple algorithm.

Since examples cannot be used, a poker bot will never run into errors in an example data set. The problem that a poker bot could run into would be the rules provided about the background information, that is, the classification rules that determine the strategy or what type of player an opponent is could be incorrect. As explained before, feedback would provide very little in the way of a solution to a set of rules that were incorrect, whether for a strategy or for classifying an opponent.

The noisy data provided by the non-deterministic opponents means many machine-learning techniques, such as decision trees, are inappropriate because of their lack of robustness in the face of errors.

Proposed strategy

Not having any examples, or the opportunity for giving feedback to the bot, the background information has to be concentrated upon. This will involve giving the bot a set of rules for grading each player, and a decision list with a set of classification rules to decide how to act in any given situation.

The machine learning would simply be a reactive strategy, that is, collecting information on a player, and reacting to their actions based on the action they have performed and the description of the type of player that the opponent modeller gives. Any feedback would have to be manually given, i.e. observing how the bot performs, or analysing the log files to spot weakness. The strategy or opponent modelling rules would then have to be revised accordingly by adjusting the hard code, normally to better fit in with human strategy suggested in no limit poker.

Section 5: Design of protocol

5.1 OVERALL CONSIDERATIONS

The overall protocol is based on the Alberta Poker Research Groups' protocol for Limit Texas Hold 'em games. They attempt to build the best Limit Poker Bot possible, and have set up an online server that anyone wishing to build an online Computer Bot can connect to and test. They have placed most of their source code online, allowing people to easily create a Computer Bot of their own, and also allowing for an insight of how they tackled certain problems relating to the set up and evaluation of the game and the server. In fact, it was hoped that their code could be almost exclusively used with just a few tweaks to turn it into a No Limit server. After several weeks of attempting to do just this, it was decided that a fresh start was needed, incorporating many of the ways that the Group had decided to solve the challenges presented. As their source code was all in Java, it made sense to use Java for the development of the server and protocol. In this way, all classes that were essential to the message passing and the handling of the game logic such as hand evaluation could be used.

Why tournament?

Alberta, in evaluating only Limit Bots, decided on an evaluation system that consisted of a starting bankroll, and then continuously playing the bots using the *same* bankroll. If the bankroll goes negative, betting is still allowed, and since the amount is limited, the play could remain sensible.
This does not apply to No Limit Poker, since there is no limit, a player could bet to a bankroll as negative as they wished, and this does not accurately represent the way that Poker is played. For this reason, a tournament structure was decided on. In this way, ranking is still possible, and the real world play of attempting to find the best No Limit Poker player is accurately emulated.

Why network and not simulation/local?

A major consideration for the project is being able to test the bots against a wide variety of players. If the poker bot were to simply be tested on a local program simulation [14], any bots or players would have to be written specifically into the software. With a server and client system, any client that conforms to the server protocol can log in and play. In this way, other people attempting to write bots or even human players can be used to test the bots against a range of players.

5.2 POKER SERVER

The task of evaluating a poker agent is difficult but crucial to determining the strengths and weaknesses of the strategies it employs, and thereby identifying possible improvements. In order the facilitate this, a server program was developed which hosts a poker game and allows multiple agents to participate by communication over a network.

It was necessary to develop the majority of the server since, at the time, there were no other poker servers that supported No Limit Texas Holdem and had a publicly available communication protocol. Myself, Steve Spencer and Claire Furney, undertook the task of designing, developing and testing the server and client jointly.

Design

The server was designed to allow players to compete in No Limit Texas Holdem tournaments.

A tournament begins with a number of players being placed in a game with an equal amount of starting money. No other players are allowed to join the tournament after it has commenced. Players play successive games of poker according to standard rules.

In accordance with the standard rules of poker tournaments the forced bets, or blinds, are gradually raised during the tournament. This is in order to avoid tournaments taking an excessively long time since if the forced bets are high; players have no choice but to risk larger amounts of money. A tournament only ends when all players but one have been 'knocked out' by losing all their money.

When a client connects to the server it must first log in, providing a user name and password. The user name is used as a unique identifier for each player and the password is checked against a stored version of each player's password to verify a user's identity. After a client has successfully logged in, it is held in a queue until it can be placed in a tournament. When there are a sufficient number of players waiting in the queue a new tournament is started in which the players are entered. Multiple tournaments can be held simultaneously in order to reduce time spent by clients waiting to start. For the same reason, players are allowed to play in a tournament if they have been 'knocked out' of another tournament that is still ongoing. If a player takes excessively long to reply to a request from the server, that player is 'frozen' and the connection to the player is terminated. During the time a player is frozen they will be made to meet the forced bets when required but will otherwise be counted as having folded their hand. If a player logs in under the user name of a player who is frozen in a tournament, they will be returned to that tournament and allowed to play normally as before. When a player has no money left, they are removed from the tournament and replaced in the waiting players queue.

The final results of tournaments are saved to disk and used to produce a player rating - an informal measure of a player's success. The full proceedings of all tournaments are also logged to a text file, intended to be used to acquire more detailed performance data on players.

Implementation

The decision was made to implement the server in Java, together with a simple graphical client to be used by human players. The choice of language was made for a variety of reasons; Java has the advantage that it has strong built in networking and multithreading libraries and also that it is capable of being run on multiple platforms with minimal effort. However a major reason was to allow the use of a Java package

called poker written by Aaron Davidson and released by the University of Alberta Computer Poker Research Group [Alberta Poker Group].

This package contains classes to represent and manipulate cards in tasks such as drawing cards from a deck or evaluating the best poker hand from a selection of cards. The package also contains several classes for sending and receiving the network messages used in a poker game. These classes helped greatly in reducing the development time of the server.

The remaining code is in a package called NLPoker, which imports many classes from poker. The NLPoker package contains classes dealing with the game logic and the package NLPoker.online contains classes concerned with message passing between server and client, scheduling games and the storing of client account information.

The server represents clients as an interface IMessageHandler to decouple the game logic from the method of communicating with clients. Currently, the only implementation of this interface uses TCP/IP message passing to communicate with clients according to protocol described below. However the interface could also be implemented to allow the use of local clients or alternative protocols. After a client connects, a separate thread handles all messages sent to and received from the client in order to improve server responsiveness. While this solution is adequate for the usage expected, it should be noted that the use of one thread per connected client does not scale to larger numbers of connected clients.

The Dealer class is responsible for administrating a poker tournament and following the standard rules for No Limit Texas Holdem. It contains a significant part of the game logic and keeps players informed of the progress of the game through the message passing interface. The GameState class is contained in Dealer and represents the current state of a game. Due to the complexity of betting in No Limit Texas Holdem with regard to side pots, a Pot class was created which takes care of side pots and allows players to represent bets as a single number, simplifying the logic needed in other areas of the code.

Communication Protocol

The initial communication protocol used was a slightly modified version of the Fixed Limit Texas Holdem protocol published by the University of Alberta. However, during development, a protocol was published as a part of the 'PokerBot World Series', a competition held by the International Conference on Cognitive Modelling 2004[ICCM 2004]. Although this protocol also used the University of Alberta protocol as a guide, it differed substantially from it. After this publication, the decision was made to change the protocol to closely match that of the ICCM. Although the protocol uses a different means of communication, using TCP/IP byte streams rather than text messages, it was intended to be easily adapted to the specification published by ICCM in order to make the task of entering subsequent competitions easy.

5.3 CLIENT

The client was developed for the sole purpose of allowing human to play against a table of bots and/or humans. Initially a text-based client was created in a style similar to the text-based client from the University of Alberta's Limit Poker site. It was decided that although this was adequate, in order to appease our supervisor, a much nicer looking graphical interface was implemented in Java using the SWING library.

Figure 5.0.1

5.4 COMPUTER BOT

The base considerations of a Computer Bot are as follows: -

- The bot must conform to the message passing protocol outlined in the server section
- The bot must reply to a request for an action within forty seconds
- The bot must be well behaved (although the server does deal with all errors discovered so far, there can be no reassurance that all possible errors have been considered)

Further specifications of the bot are up to the programmer, and all details of this project's implementations are outlined below.

Section 6: Testing & Benchmarking

6.1 BENCHMARKING

Firstly, a consistent structure of play needed to be chosen. As discussed in the protocol section, a tournament style was chosen, as opposed to continuous play with the same bankroll. The parameters of the tournament were taken from an online poker tournament, so as to replicate as accurately as possible real world no limit poker play. Multiple tournaments were to be played between the Computer Bot and various other bots/humans, with the overall percentage of tournament wins giving a good idea as to how well the bot plays.

Benchmarking a bot in no limit poker is quite a difficult task in itself. Game dynamics can change drastically between different tables, depending on how many other players there are, and what type each player is at the table. With this in mind, some simple tests can indicate whether the bot can survive in a more complex situation. For this, the services of other members of the bot community have been called into action.

6.2 ALWAYS CALL BOT

As the name implies, this bot will call in any situation. Over enough time, one Always Call Bot playing against another Always Call Bot will give a win percentage of roughly 50% for each bot. Although human players can beat the Always Call Bot very easily, it can provide information as to whether or not the Computer Bot has a grasp of when it has a good poker hand.

6.3 ALWAYS RAISE BOT

This bot raises a random amount *every* time it is required to perform an action. Over enough time, one Always Raise Bot playing against another Always Raise Bot will give a win percentage of roughly 50% for each bot. An Always Raise Bot is very difficult to play against, even for human players. The Always Raise Bot will give an indication as to how well a Computer Bot can react to its opponents' actions.

6.4 RANDOM BOT

Again, as the name suggests, this bot will perform a random action every time it is required to perform an action. When this action is a raise, it will raise a random amount. A Random Bot can give an idea about both whether an Computer Bot has a grasp of when it has a good poker hand, and also whether it can model opponents well.

6.5 INITIAL TESTS

To get an initial indication of how good a Computer Bot is, a series of heads up tournaments are to be played against each type of bot. The total amount of tournaments should be large enough to negate any lucky runs of cards; so 100 tournaments should be adequate. In particular, the results against the Always Call Bot and the Always Raise Bot are to be paid attention to.

Section 7 : Outline of implementations, hypotheses, and analysis of testing

7.1 GENERAL OUTLINE OF BOTS' IMPLEMENTATION

Implementation common to all bots

The following figures (7.0.1 and 7.0.2) show the basic structure of how the classes within the bot work. Because the server handles all of the game logic, the current state class within each bot holds all pertinent information that the server has passed to the bot that deals with the current game state. The hand evaluator, common to all bots, is explained in detail below. Each bot acts upon the recommendation of the action giver class contained within.

Implementation of bots with no opponent modelling

The action giver acts solely upon the information supplied by the current state class and the hand evaluator class.

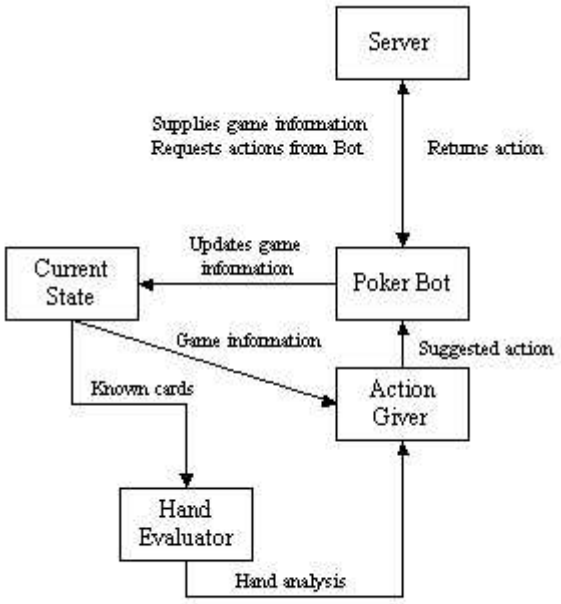


Figure 7.0.1

Implementation of bots with opponent modelling

The action giver acts upon information supplied by the current state class, the hand evaluator class, and the opponent modeller class. The opponent modeller class both loads and saves information to the stored opponent information class.

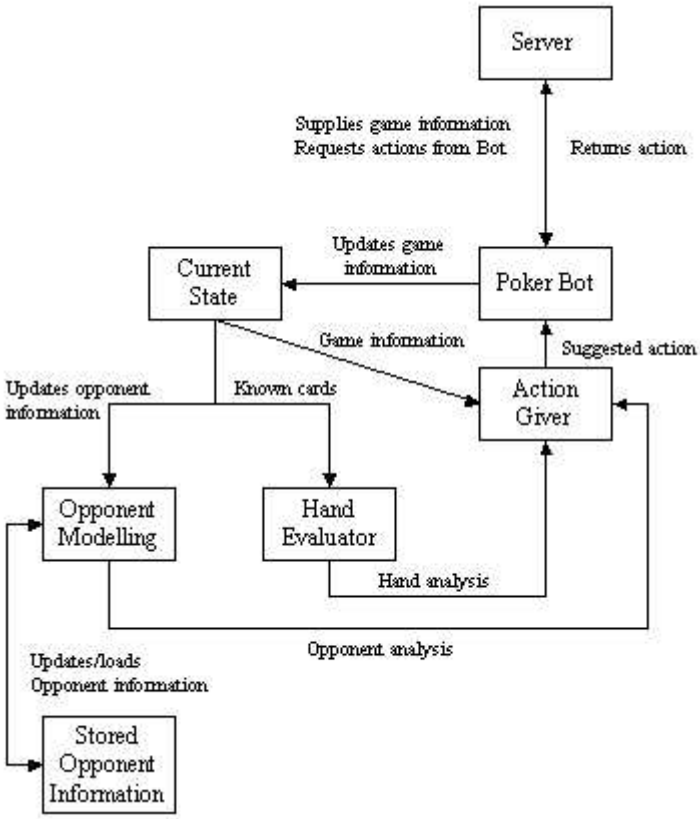


Figure 7.0.2

7.2 HAND EVALUATION

Overview

Much thought had to be given to the subject of how to evaluate hands.

Firstly, a method had to be devised by which a hand can be compared to another and the strongest hand determined.

Secondly, a method had to be developed that would give a grading to the hand that was input together with the number of opponents.

Implementation

Since the bots were all implemented in Java, the Java classes written by the University of Alberta computer research group [15] for hand evaluation could be used.

The first problem was solved by a hand ranking function and a sorting function (in order to determine the best rank for a 5 card hand from a 7 card input), both taken from the University of Alberta's implementation. The hand ranking function takes any 5-card hand and returns an integer unique to that hand. If hand 1 has a rank higher than hand 2, hand 1 will beat hand 2 in a showdown.

The second problem required developing a function that takes the pair of hole cards, along with the known board cards and the amount of opponents, and returns a percentage of showdowns that those hole cards would be expected to win.

The function developed takes as an input the hole cards, the known board cards and the amount of opponents still in the game. Then, if the total amount of board cards known is not 5, it will select random cards from the deck until 5 board cards are known. Then, it will select a pair of random cards from the deck for every opponent in the game. Finally, each opponent hand is compared to the input hand. If none of these hands are ranked higher, the input hand is deemed to have won. This process is then repeated 10000 times, and the percentage of hands won by the input hand is returned.

7.3 OPPONENT STATISTICS

Overview

The challenge with opponent statistics was how to keep the most up to date statistics in a saved file whilst only having the statistics of the opponents in the current game available to the bot.

Implementation

Every time a new game starts, the bot writes all information that it currently knows about the players in the previous game to file. Then, it collects from this file the statistics of the players in this new game. In this way, the saved file is kept up to date, and the opponents in the game are the only statistics that the bot keeps in the current memory.

For reference in the following sections, the term rating is defined

Def rating: The amount of standard deviations a hands' rank is above the mean of the current distribution

7.4 OPPONENT MODELLING

Within the class of opponent statistics, there exist functions that return important information about that opponent. These include: -

GetAggression (): - returns the amount of times that the player has bet over the total amount of actions performed

GetGroup (onblind): - takes a Boolean as to whether the current player is on the blind, and returns the average Sklansky group of the player's hole cards in this situation

GetAverageRank (): - returns the average rank of the hands that the player has taken to the showdown

GetRating (stage, action): - takes the current stage (preflop, flop, turn, or river) and the last action performed (check, call, or raise) and returns the player's rating given these parameters

GetProb (stage, action, amount): - takes the current stage, the last action performed and the amount called or raised, and returns the estimated probability of the player's hand winning given these parameters

7.5 FUNCTIONS USED IN THE BOTS

GetProb () – performs a roll out of possible opponents hands, and returns the percentage of times that the given hand is expected to win

GetLowestGroup () – analyses information about opponents still in the game, and returns the lowest Sklansky group of all opponents in the game given their current situation

GetRating () – returns the rating of the bots' current hand

GetHighestRating () – returns the highest average rating of all other players still in the game given the actions that they have already performed in the game

GetHighestRank () – returns the highest average rank of hands that went to the showdown of all players still in the game

GetHighestProb () – returns the highest estimated probability of opponents still in the game, based on their actions

7.6.1 1st IMPLEMENTATION

The first implementation is a bot that only attempts to recognise how good its hand is. This is essentially to make sure that all functions being used for hand evaluation are working properly, and that a general idea of how good a hand is can be determined. There is no opponent modelling in this bot.

Decision tree for the first implementation

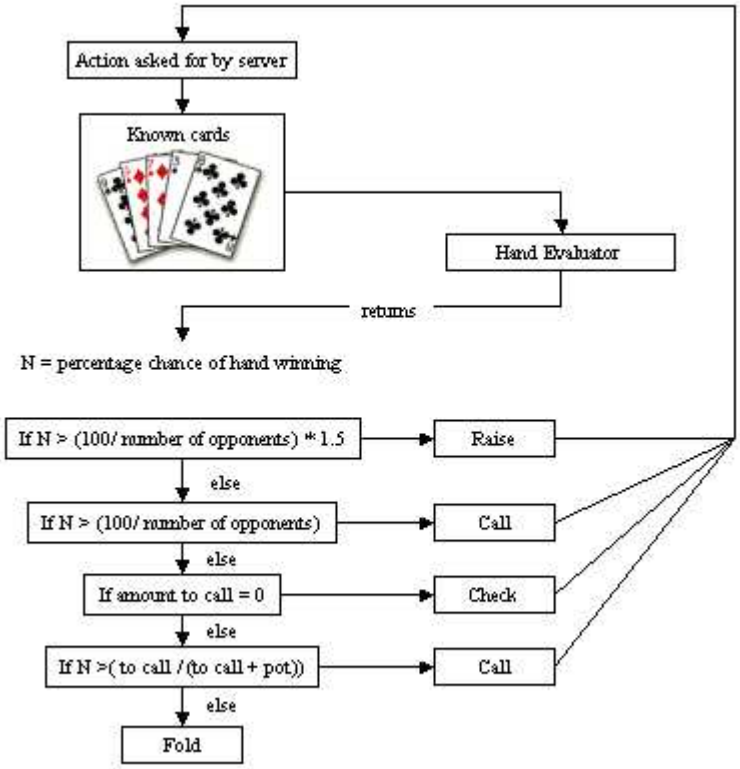


Figure 7.1.1

The decision list and classification rules for the action giver contained in the first implementation:

All stages

The predicted winning percentage is return from the call to the hand evaluator. The average percentage of times that the bot will win is found out by dividing the amount of players still in the pot into 100.

-If the predicted percentage is 1.5 times the average percentage, the bot will raise. This raise is determined by calling RaiseFunction ().

-If the bot doesn't raise, if the pot odds are higher than the chance of winning, the bot will call.

-Otherwise the bot will fold

RaiseFunction ():

$$\frac{\text{Predicted probability of winning} - (1.5 * \text{average probability of winning})}{100 - (1.5 * \text{average probability of winning})} * \text{Bankroll}$$

7.7.1 HYPOTHESIS

Using only hand recognition, this bot should out perform AlwaysCallBot and RandomBot.

7.8.1TEST RESULTS

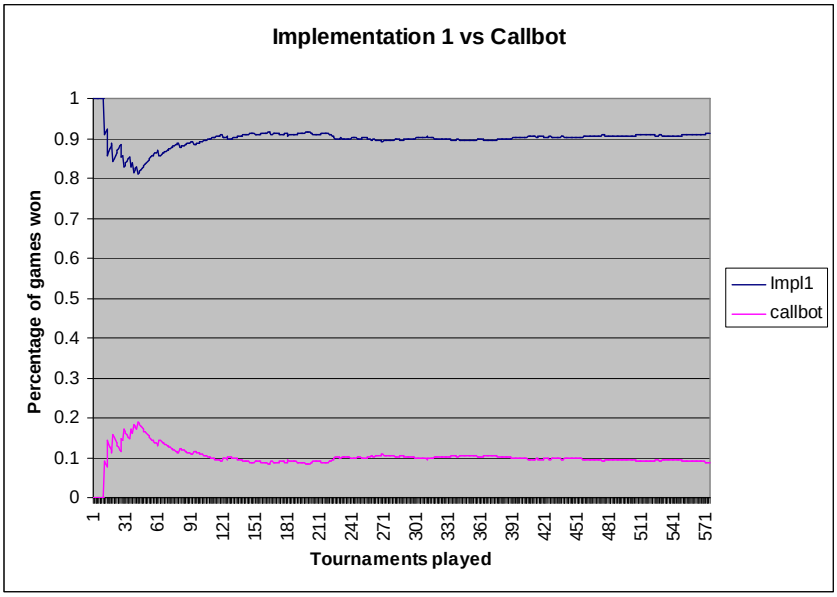


Figure 7.1.2

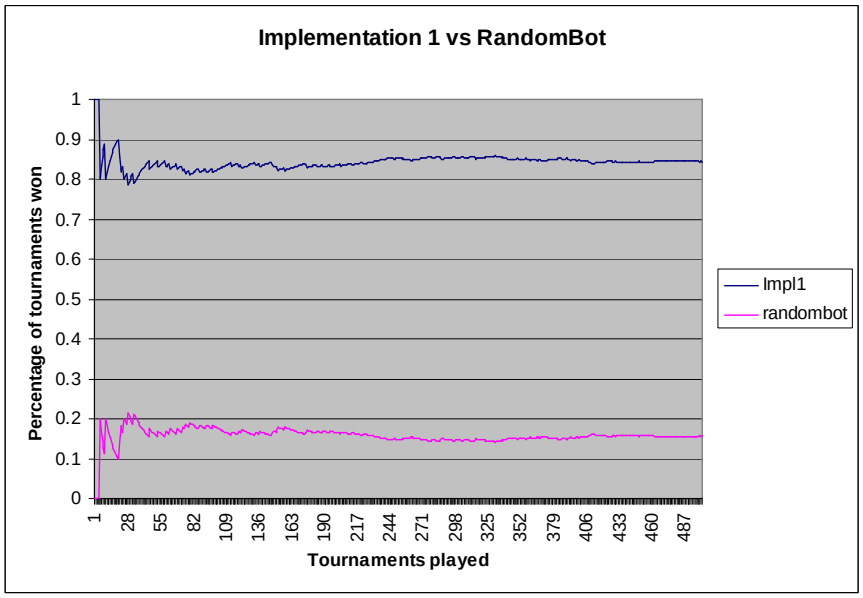


Figure 7.1.3

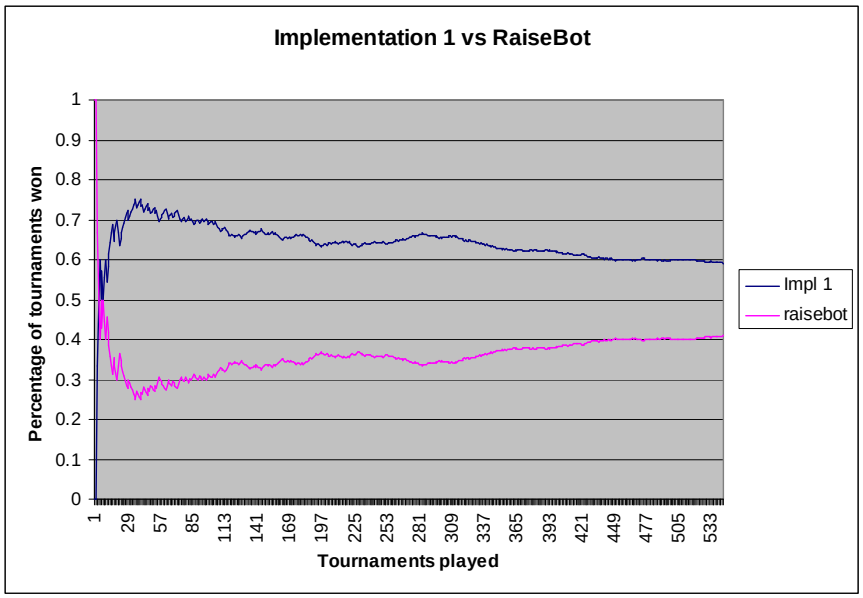


Figure 7.1.4

7.9.1 ANALYSIS OF RESULTS

Figures 7.1.2 – 7.1.4 shows that Implementation 1 fares well against all of the test bots.

7.10.1 OVERALL

In relation to the aims of the project, several of the cognitive activities have been achieved, to varying degrees of satisfaction.

1) The simplistic assumption that opponents only receive average cards and any hand over the average probability being played provides reasoning under uncertainty to a limited extent. The results against both AlwaysCallBot and RandomBot provide evidence that this is adequate, at the very least against non-aggressive players.
2) The rollout simulation becomes more accurate as the amount of potential situations evaluated is increased. With 10000 potential hands being tested at every action request, it can be said that the bot deals very well with probabilistic outcomes, and this has been borne out in the tests against AlwaysCallBot and RandomBot.
3) The decision making action of how much money to bet is handled by a very simple function, where a hand that has a good probability of winning has more money bet on it, but is quite difficult to evaluate against the bots, since they have no recognition of other players. A human or a learning bot looking for the correct information would easily be able to tell when the 1st Implementation has a good hand or not.
4) The bot does not understand different styles of play. It only looks at how much it has to bet in order to stay in the hand, and does not take into account how realistic the bet made by some other player could be.

5) The bot does not understand different inferences of intent from opponents' bets; it simply assumes that a higher bet made means a better hand.
6) The bot does not understand deception – this is evidenced by the results against the AlwaysRaiseBot as compared to the other two bots. Although Implementation 1 still has a winning record against this bot, it is hoped that further implementations, which will incorporate opponent modelling, will fare even better.
7) There is no pattern recognition in the bot.
8) The bot does not factor the impact of the amount of bets that it makes, and offers no deception of its own. The bot literally plays hands that it is likely to win, and doesn't play hands that it isn't likely to win, with the exception of when there is nothing to call (i.e. check to showdown)

7.11.1 POSSIBLE REFINEMENTS

Opponent modelling needs to be introduced in order to make sure that the bot plays hands that it is most likely to win. Actions taken might also be broken down into different strategies at different stages of the game, to allow for greater refinement of the bot for the game, and to better mirror human strategy.

7.6.2 2nd IMPLEMENTATION

The second implementation attempts to build upon the success encountered by the first implementation. In this case, some opponent modelling has been added in order to take into account aggressiveness of other players. An opponent information collecting class is included, which is updated based on what the opponents have done. Categories include the total amount of each action performed, an overview of where the player folded percentage wise, the hole cards that the player went to the showdown with, and the hand rank of the hand that the player went to the showdown with.

Opponent modelling structure

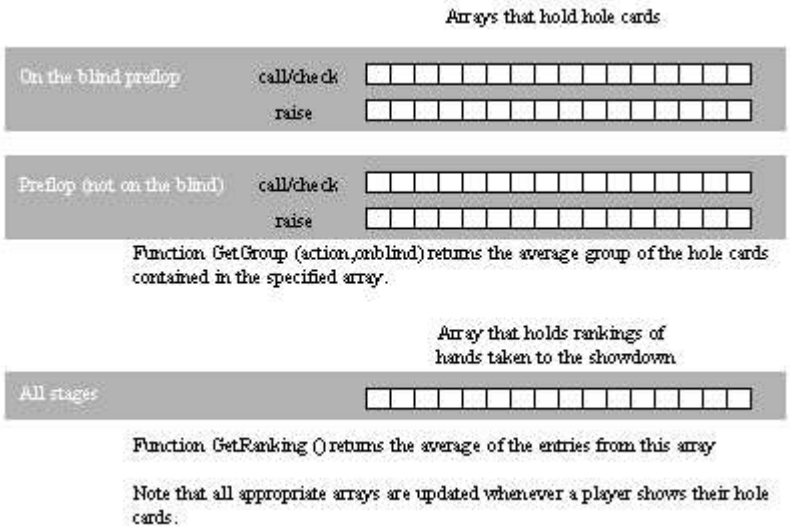


Figure 7.2.1

Decision tree for the first implementation

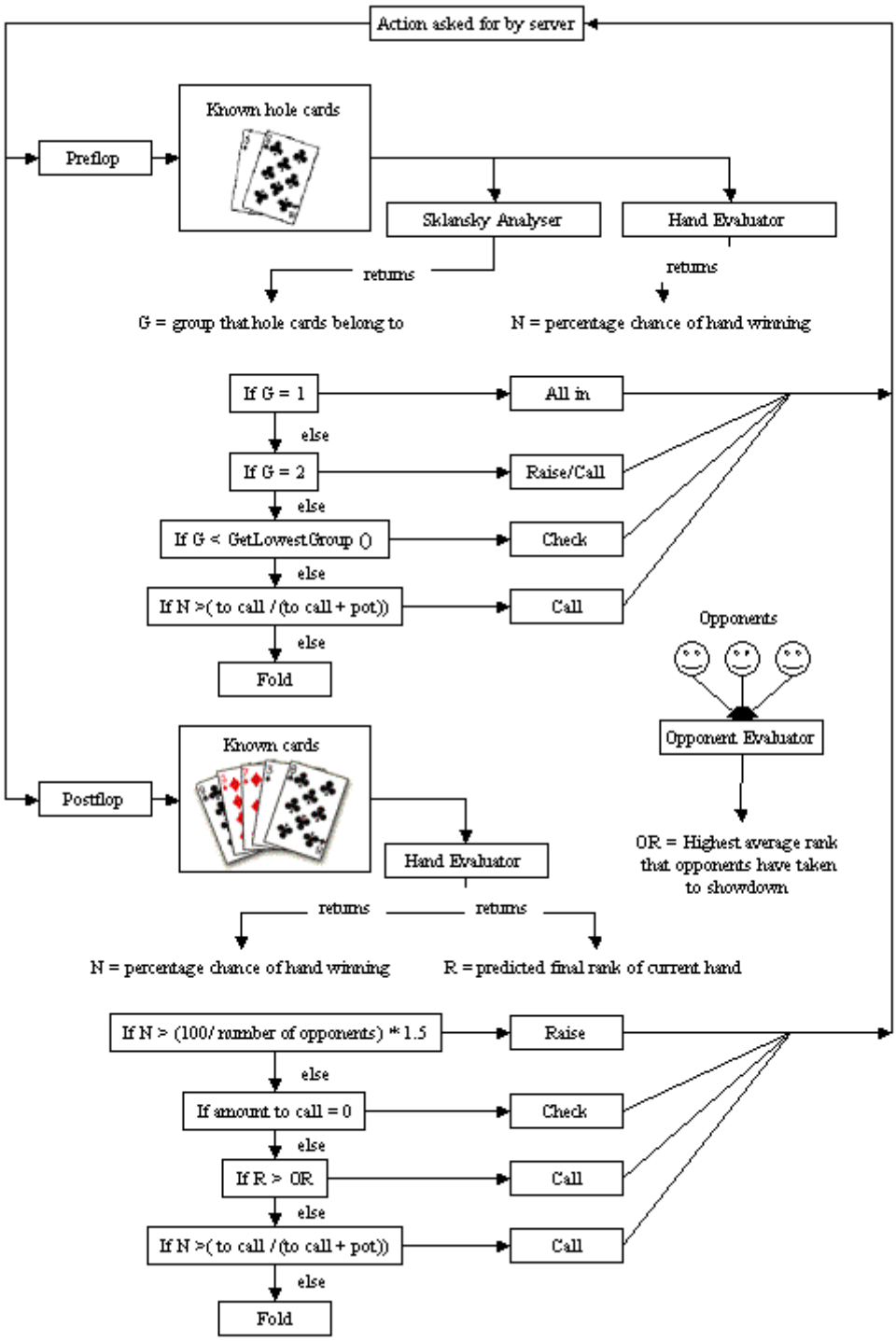


Figure 7.2.2

The decision list and classification rules for the action giver contained in the second implementation:

For this implementation, GetLowestGroup () and GetHighestRank () methods are used.

In order to better mirror human strategy, different functions are used at different stages of the game. The rules for acting at each stage are outlined below:

Preflop

Firstly, the bot analyses its hole cards and finds out the group that they are in.

-If the group is 1, it will go all in.

-If the group is 2, it will raise up to half of its bankroll, and call the rest of the way.

-If the group is 3 or higher, the bot calls the GetLowestGroup () function. If the bots' group is lower than this, and the amount to call is 0, it will call.

-Otherwise, GetProb () is called, and if the probability of winning is lower than the pot odds, then the bot will call, else it will fold.

Post Flop

Firstly, the bot calls the GetProb () function.

-If the predicted winning percentage is 1.5 times the average, the bot will raise. This raise is determined by calling RaiseFunction ().

-If the amount to call is 0, then the bot will check

At this point, the bot calls the GetHighestRank () function.

-If the bot has a higher ranked hand than any of the opponents, it will call

-Otherwise, if the probability of winning is lower than the pot odds, then the bot will call, else it will fold

RaiseFunction ():

$$\frac{\text{Predicted probability of winning} - (1.5 * \text{average probability of winning})}{100 - (1.5 * \text{average probability of winning})} * \text{Bankroll}$$

7.7.2 HYPOTHESIS

This bot should play well against both the Always Call and the Always Raise bots.

7.8.2 TEST RESULTS

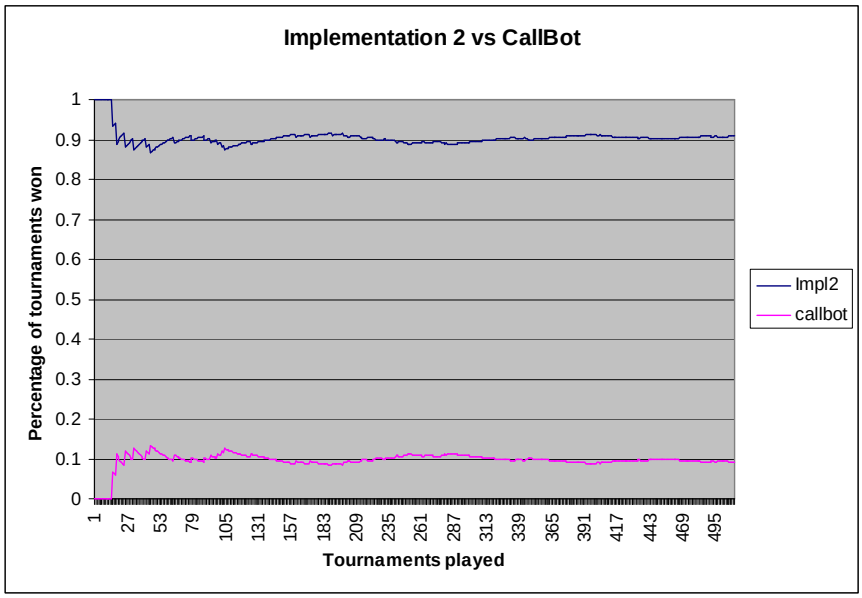


Figure 7.2.3

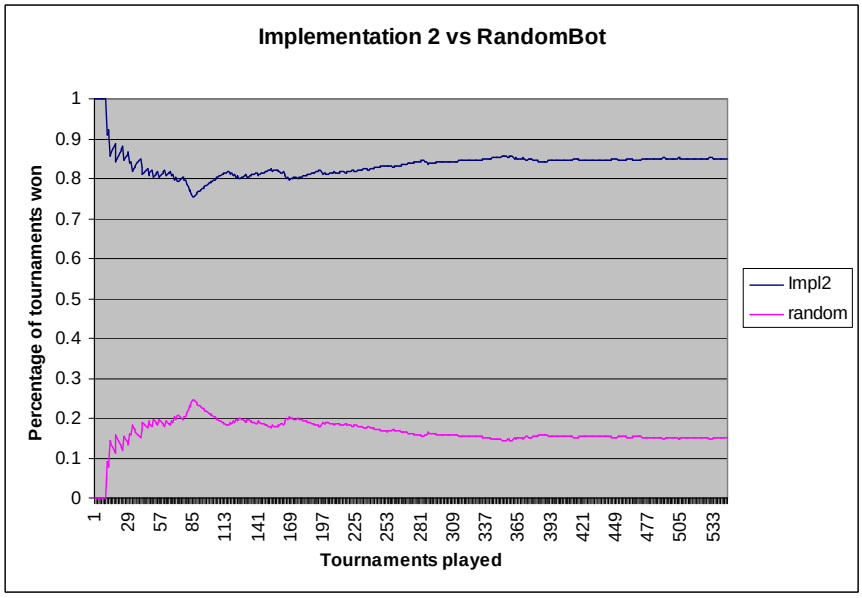


Figure 7.2.4

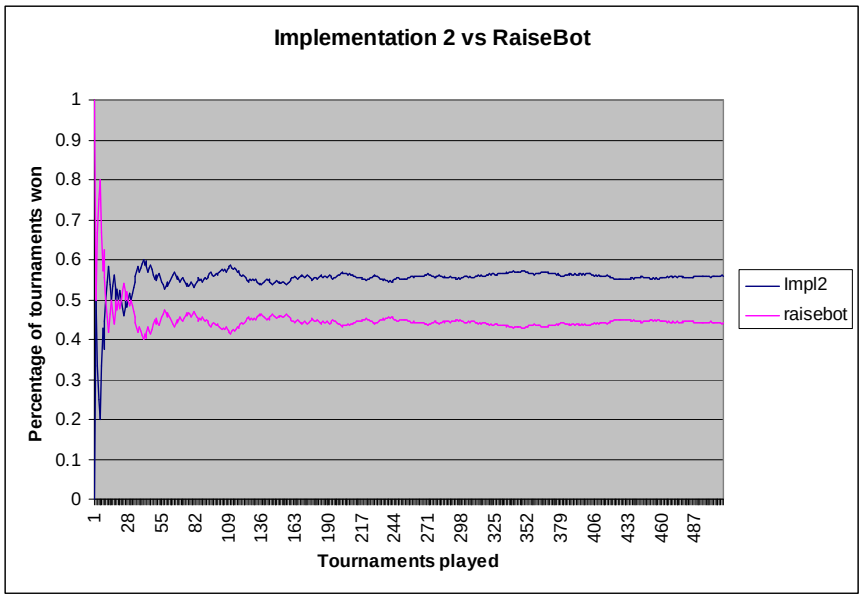


Figure 7.2.5

7.9.2 ANALYSIS OF RESULTS

Figures 7.2.3 and 7.2.4 show that the hand evaluation within the implementation is just as good as in Implementation 1. Figure 7.2.5 shows a slight decrease in the percentage of tournaments won when compared to Figure 7.1.4. This could simply be due to a run of results that means the opponent modelling misevaluates the opponent, or it could mean that the underlying idea behind the way in which the opponent is evaluated is incorrect. A positive to be pulled from the test illustrated in 7.2.5 is the fact that the bot can be seen to be learning against the aggressive strategy of the opponent.

7.10.2 OVERALL

1) The bot can reason under uncertainty, with any hand over the opponents' estimated best hand rank being played. There is an assumption that the opponents don't change their style, and there is no recognition to give a better idea as to what type of hand an opponent holds when they perform a particular action.
2) The bot deals very well with probabilistic outcomes, with the rollout simulation becoming more accurate as the amount of potential situations is evaluated. 10000 potential hands are used here, and this has proved to be accurate enough in the above tests.
3) The decision making action of how much money to bet is handled by a very simple function, where a hand that has a good probability of winning has more money bet on it, but appears to be quite sound. A human or a learning bot looking for the correct information would easily be able to tell when Implementation 2 has a good hand or not. Therefore this point has been covered adequately, but not excellently.
4) The bot understands different styles of play to an extent. There is no specific action opponent modelling, but a very basic idea of how an opponent plays is gathered from the average rank that they went to the showdown with, and also what hole cards they went to the flop with.

5) The bot does not understand different inferences of intent from opponents' bets; it simply assumes that a higher bet made means a better hand.
6) The bot does not understand deception.
7) There is no pattern recognition in the bot.
8) The bot does not factor the impact of the amount of bets that it makes, and offers no deception of its own. The bot literally plays hands that it is likely to win, and doesn't play hands that it isn't likely to win, with the exception of when there is nothing to call (i.e. check to showdown)

7.11.2 POSSIBLE REFINEMENTS

The opponent modelling in this implementation is possibly inadequate – using maximum average rank of hand gives no real indication as to what hand rank a given player would have on a given flop. A system needs to be set up that factors in the type of actions performed at different stages in the game, and how the board cards affect these bets.

7.6.3 3rd IMPLEMENTATION

The third implementation attempts to build upon the success of the previous implementation.

A possible extension centres on the reaction to opponents hands, particularly the context in which the player's hands have been taken to showdown. For instance, a flop might not allow some certain hands to be possible, and the turn and the river further narrow down the possibilities. With this in mind, it is important to have a system that will give an indication under which circumstances a particular ranked hand was taken to the showdown.

There are several problems facing the collection of this type of data:

The first problem is finding the potential of a board. There are many possible hands from any given situation, the sheer volume of which prohibits an exhaustive search. The proposed solution is to do a roll out of possible hands from each given stage, taking note of the mean for the average hand rank, and the standard deviation of the other ranks from this mean to give an idea of the spread of possible hands.

The second problem is grading a given hand against the distribution. The proposed solution for this is to find out how many standard deviations above the mean an opposing player performed a particular action. This ratio will be referred to as a rating henceforth.

The way that these data are collected is again from showdowns. As soon as the player has shown their hole cards, their potential hand rank at each stage is calculated against the potential distribution, and the results stored based on what action was performed by the player at that stage. In this way, 6 arrays are built up, one for calling on the flop, one for raising on the flop, etc… From these, an average rating for a player's actions at each stage of the game can be calculated.

On starting the bot, there is also a parameter for inputting an aggression figure, between 0 and 100. In this way, given enough time, many of the 3rd Implementations' bots can be played against each other, each with a different aggression, to determine the best aggression.

Opponent modelling structure

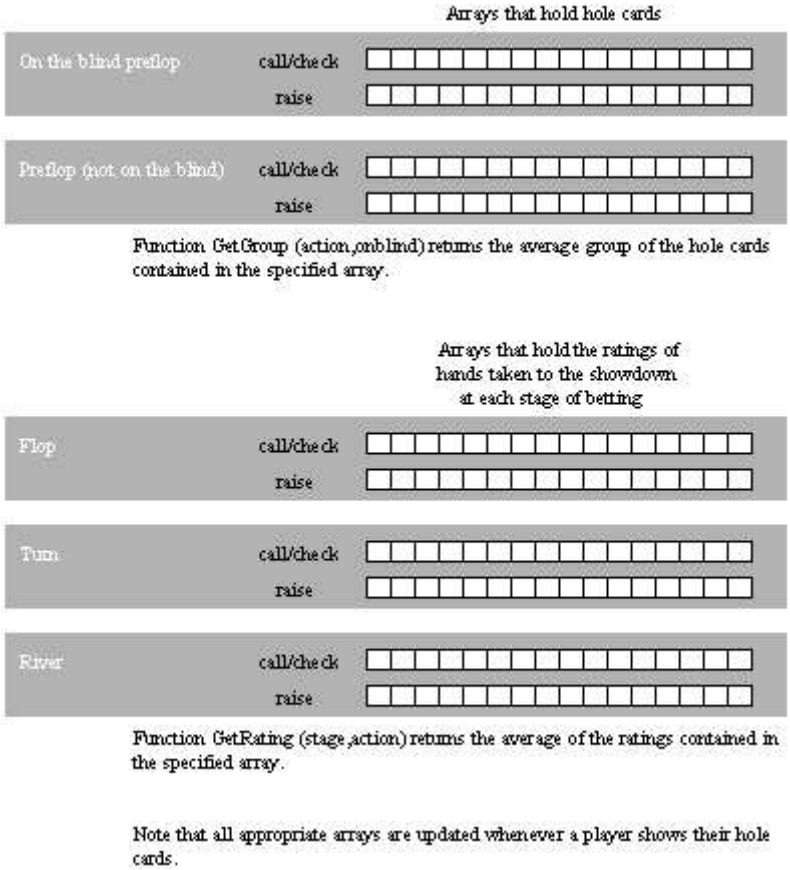


Figure 7.3.1

Decision tree for the third implementation

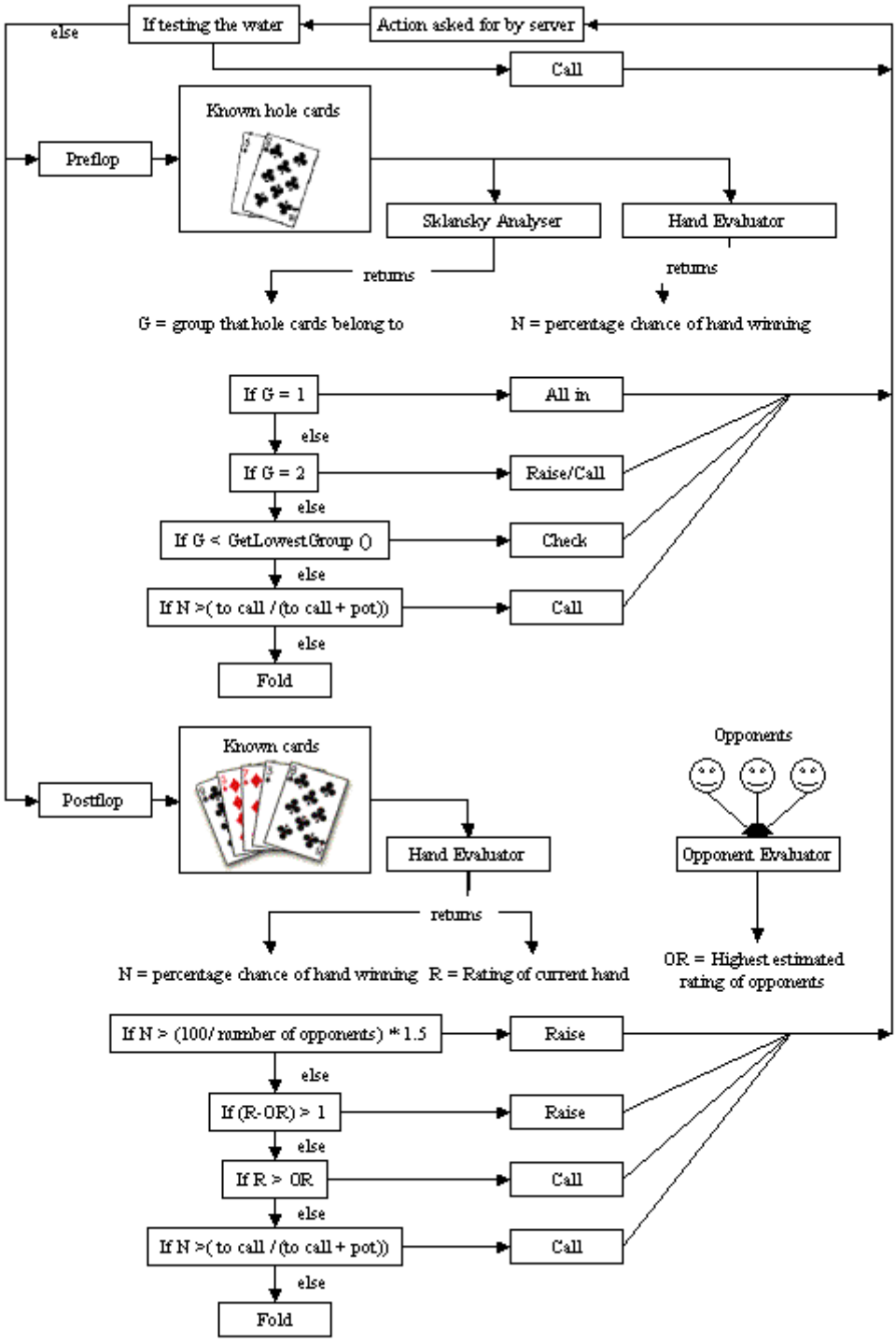


Figure 7.3.2

The decision list and classification rules for the action giver contained in the third implementation:

The overall strategy for the 3rd implementation is outlined below:

Testing the water ()

Before the first hand, a random number between 0 and 20 is chosen. After this amount of hands, the bot will call through to the showdown, regardless of what hands it has. Another number is then chosen, again at random, and the process repeated. This is to make sure that the bot is still collecting information on the other players, when statistics might otherwise tell it to fold off of most pots. In this way, "Paying for information", the bot will always have up to date information on all other players, possibly thwarting a player who starts tight, then becomes aggressive and loose and bluffing the bot off of the pot.

Preflop

Firstly, the bot analyses its hole cards and finds out the group that they are in.

-If the group is 1, it will go all in.

-If the group is 2, it will raise up to half of its bankroll, and call the rest of the way.

-If the group is 3 or higher, the bot calls the GetLowestGroup () function. If the bots' group is lower than this, and the amount to call is 0, it will call.

-Otherwise, GetProb () is called, and if the probability of winning is lower than the pot odds, then the bot will call, else it will fold.

PreflopRaiseFunction ():

$$\frac{\text{Probability of winning} - (1.5 * \text{average probability of winning})}{100 - (1.5 * \text{average probability of winning})} * \text{Bankroll}$$

Post Flop

If the bot has nothing to call, it must decide whether to raise or not.

Firstly, the bot calls the GetProb () function.

-If the winning percentage is 1.5 times the average, the bot will raise. This raise is determined by calling PreflopRaiseFunction ()

If it doesn't raise, or there is a bet to call, the bot calls the GetHighestRating () and GetRating () functions

-If the difference between the bots' rating and the GetHighestRating value is greater than 1, it will raise. This raise is determined by calling the post flop raise function.

-If the difference between the bots' rating and the GetHighestRating value is greater than 0, it will call.

-Otherwise, if the probability of winning is lower than the pot odds, then the bot will call, else it will fold.

FlopRaiseFunction ():

(Bots' rating – Highest Opponent Rating) * (aggression/200) * bankroll

TurnRaiseFunction ():

(Bots' rating – Highest Opponent Rating) * (aggression/150) * bankroll

RiverRaiseFunction ():

(Bots' rating – Highest Opponent Rating) * (aggression/100) * bankroll

Implementation

7.7.3 HYPOTHESIS

This bot should perform well against all opponents after having been given time to learn.

7.8.3 TEST RESULTS

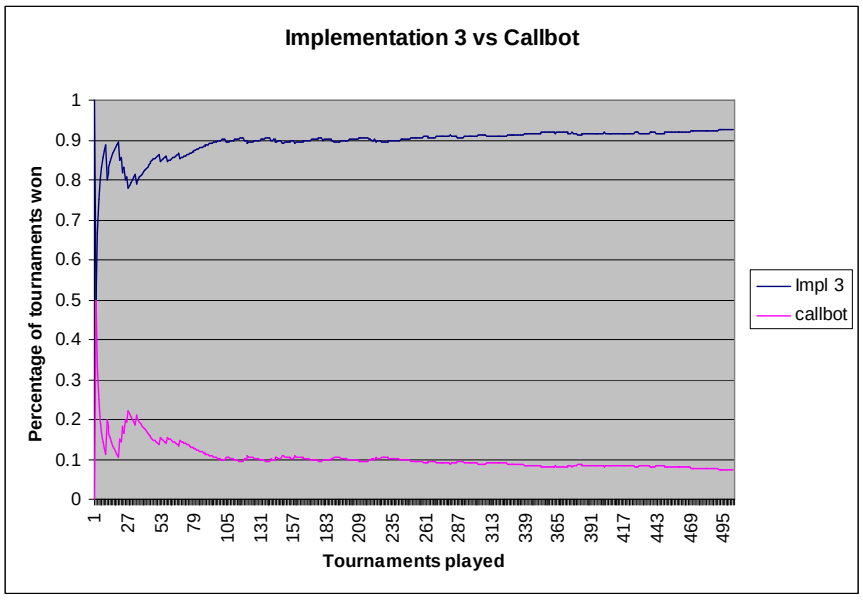


Figure 7.3.3

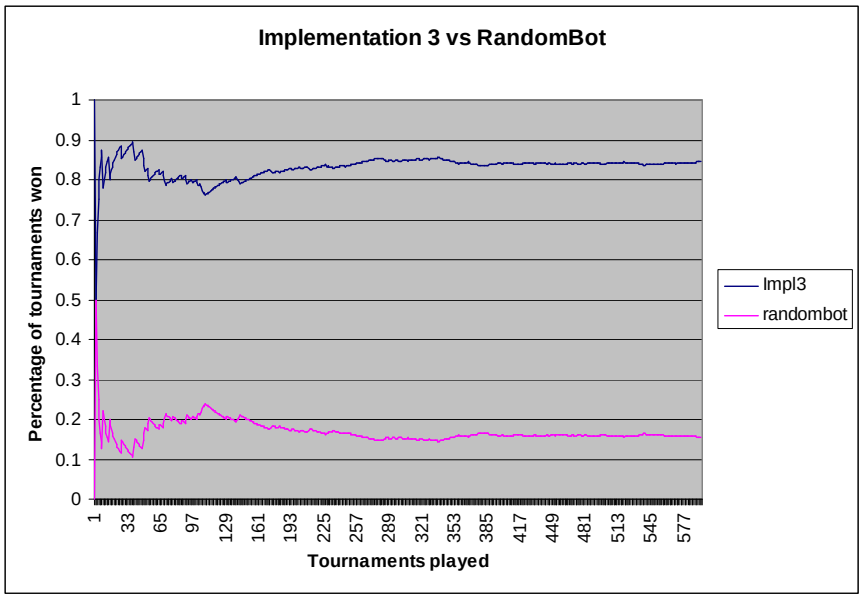


Figure 7.3.4

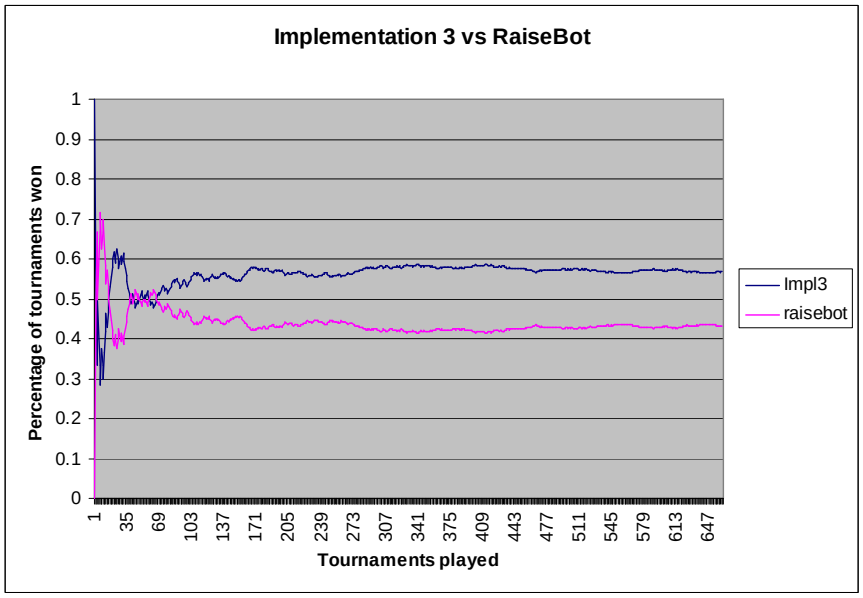


Figure 7.3.5

7.9.3 ANALYSIS OF RESULTS

Again, Figures 7.3.3 and 7.3.4 show that the underlying hand evaluation in the bot is sound. Figure 7.3.5 shows the bot learning against an aggressive player, however, the results are better than Implementation 2's versus the AlwaysRaiseBot.

7.10.3 OVERALL

1) The bot can reason under uncertainty, with any hand over the opponents' estimated best hand rank being played. There is no assumption that the opponents don't change their style, since a rolling average of hand ranks of the past 50 hands is taken for each type of action at each particular stage.

2) The bot deals very well with probabilistic outcomes, with the rollout simulation becoming more accurate as the amount of potential situations is evaluated. 10000 potential hands are used here, and this has proved to be accurate enough in the above tests.
3) The decision making action of how much money to bet is handled by a very simple function when no-one else has bet, but when someone else has bet, the amount to raise depends entirely on the bots' estimate of the best hand rank that an opponent holds. This makes it very difficult to predict whether the bot has a good hand or not based upon how much it bets.
4) The bot understands different styles of play. There is specific action opponent modelling, and whenever an opponent makes an action, an estimate of his hand rank can be made from the hand ranks held in memory.
5) The bot understands different inferences of intent from opponents' bets, simply to the extent that a certain action gives an indication as to what type of hand the opponent has now, based on the hands that they have held in the past.
6) The bot understands deception to the extent that it will play a much lower rank hand than usual against a very loose and aggressive player.
7) There is no pattern recognition in the bot, particularly in the area of reacting to the amount that the blind is, and how much money is left in the bankroll compared to this.
8) The bot does not factor the impact of the amount of bets that it makes, and offers no deception of its own, apart from when it is using the testing the water function.

7.11.3 POSSIBLE REFINEMENTS

The opponent modelling only takes into account actions performed. This means that the amount raised by the other player isn't considered, and this can give a good indication as to what type of hand they hold. The next implementation could hold some type of rating refinement that also gives weight to the amount raised. When the amount to call is zero, a magic number is still used in the raise function. This could possibly be linked to the aggression parameter. There is no consideration of the bankroll that the bot has left compared to how high the blinds are. This means that the bot can get forced out of the pot when it should possibly loosen up and attempt to take risks to get back into the game.

7.6.4 4th IMPLEMENTATION

The fourth implementation attempts to address the shortcomings of the 3rd Implementation. This includes an attempt at recognising amounts raised and called as an indication of how good a player's hand is. Also, it was noticed that the method of putting the hole cards into groups is fairly coarse – this is simply a method for human players to better remember which sets of hole cards are worth playing or not.

This implementation estimates opponent's potential hands based on rollout simulations at every stage. The bot keeps 15 arrays for each player, for the actions check, call and raise at each of the stages: preflop on blind, preflop not on blind, flop, turn, and river. For each of the check arrays, when the hole cards for that player are revealed, the probability of that hand winning is updated in the next position in the array. For the call and the raise arrays, the probability of that hand winning is divided by the amount that was called or raised in that round is updated into the array.

When the estimated probability of a players' (unseen) hand is needed; if the player has checked, the average of the check array is returned, if the player has called, the average of the call array is returned, and multiplied by the amount that that player called to give the probability, and if the player has raised, the average of the raise array is returned, and multiplied by the amount that the player has raised in order to give the probability. The GetHighestRating2 () function returns the highest of these probabilities from all opponents still in the game.

The values of the raise array are worked out as such:

$$\frac{\text{Probability of Hand winning in round}}{\text{Amount of money raised in round}} = \text{value input into array}$$

If an opponent has bet an amount, A, on the flop, say, we work out the estimated probability of his handing winning as such:

Average value of raise array on the flop * A = estimated probability of opponent's hand winning

Opponent modelling structure

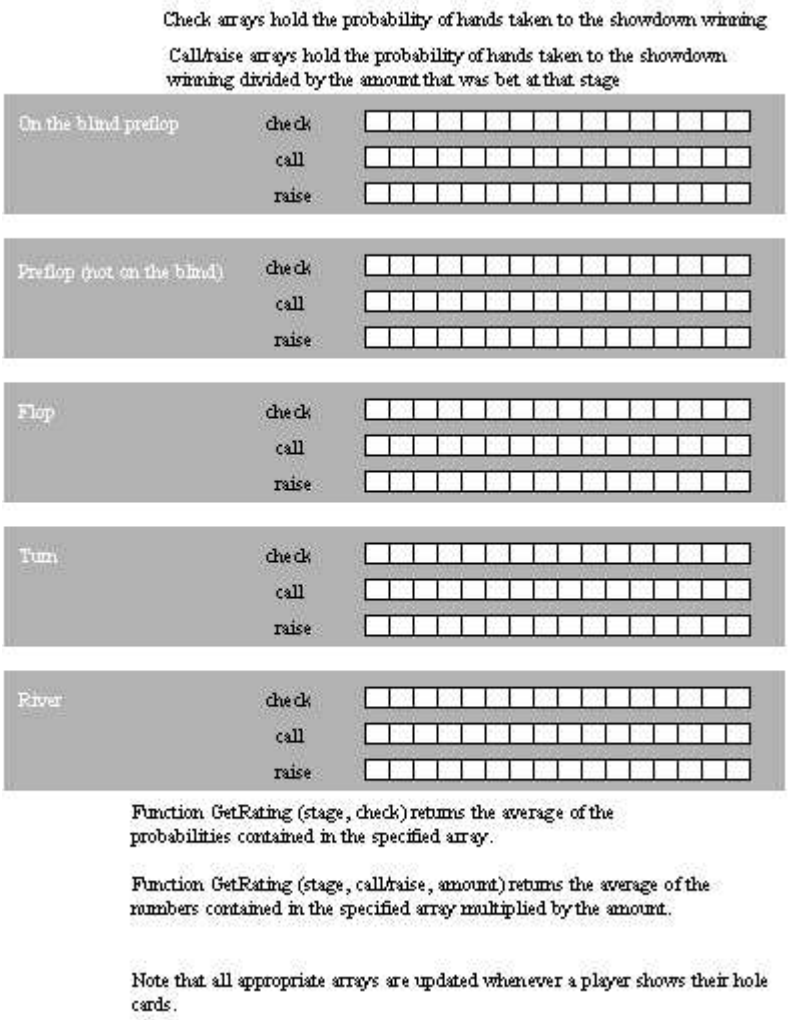


Figure 7.4.1

Decision tree for the fourth implementation

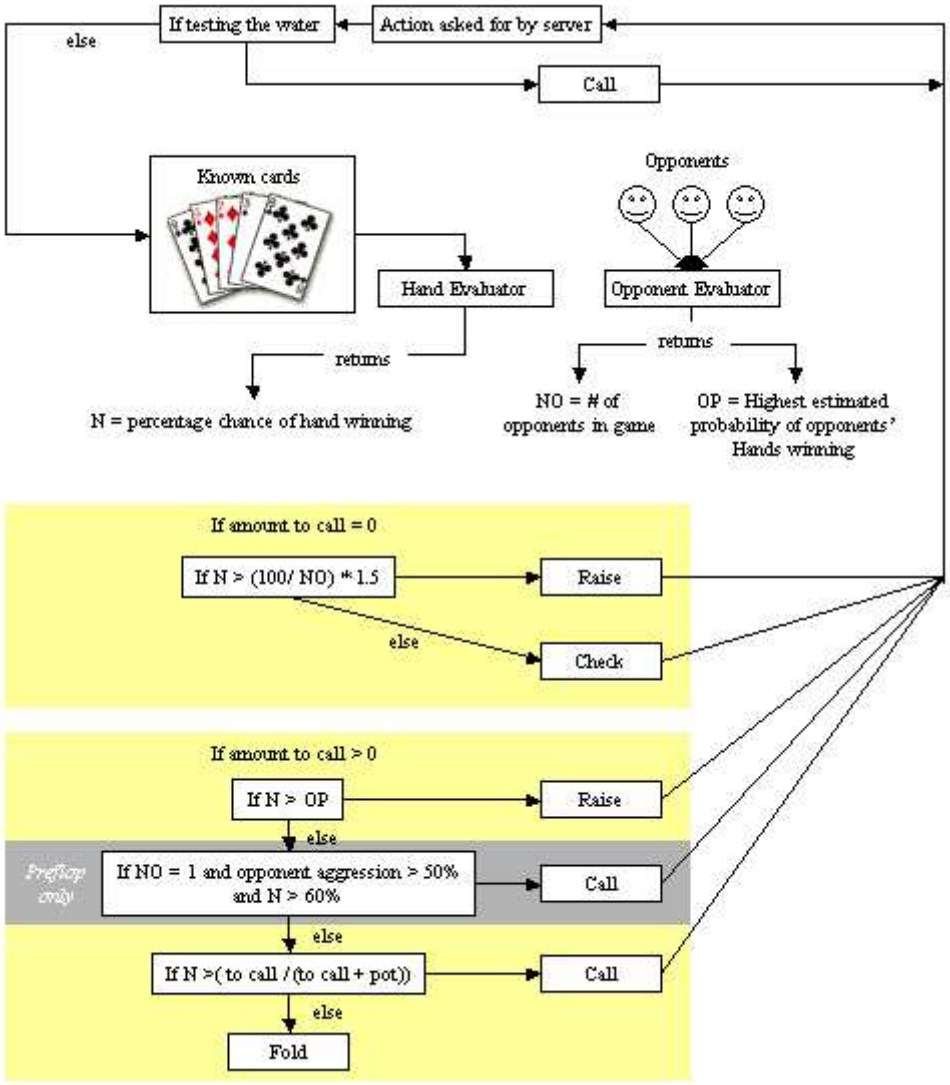


Figure 7.4.2

The decision list and classification rules for the action giver contained in the fourth implementation:

The new approach is as follows: -

Preflop

Firstly, the bot calls the GetProb () function

If there is nothing to call

- If the probability of winning is over 1.5 times the average probability of winning, then the bot will call the RaiseFunction ()

- Otherwise, the bot will check

If there is something to call, the bot calls the GetHighestRating2 () function

- If the probability of winning is higher than this, then the bot calls RaiseFunction2 ()

Then, if there is only one other player at the table, and his aggression is greater than 50%

- If the probability of winning the hand is higher than 60%, the bot will call

- Otherwise, if the probability of winning is less than the pot odds, then the bot will call, else it will fold

Postflop

Firstly, the bot calls the GetProb () function

If there is nothing to call

- If the probability of winning is over 1.5 times the average probability of winning, then the bot will call RaiseFunction ()

- Otherwise, the bot will check

If there is something to call, the bot calls the GetHighestRating2 () function

- If the probability of winning is higher than this, then the bot calls the RaiseFunction2 ().

- Otherwise, if the probability of winning is less than the pot odds, then the bot will call, else it will fold

RaiseFunction (): -

$$\frac{\text{Probability of winning} - (1.5 * \text{average probability of winning})}{100 - (1.5 * \text{average probability of winning})} * \text{Bankroll}$$

RaiseFunction2 (): -

Bankroll * aggression * (Probability of winning – Opponent's probability of winning)

Implementation

7.7.4 HYPOTHESIS

This bot should perform well against all opponents after having been given time to learn.

7.8.4 TEST RESULTS

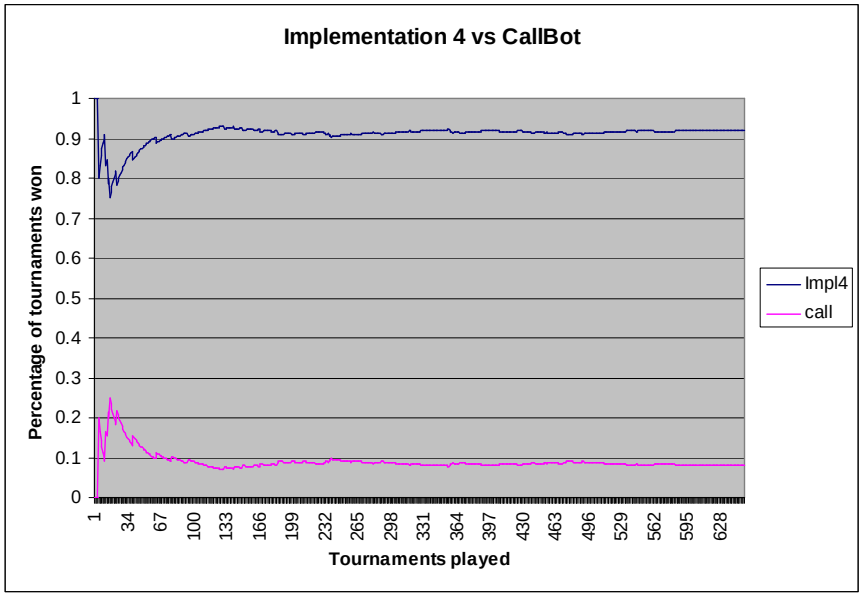


Figure 7.4.3

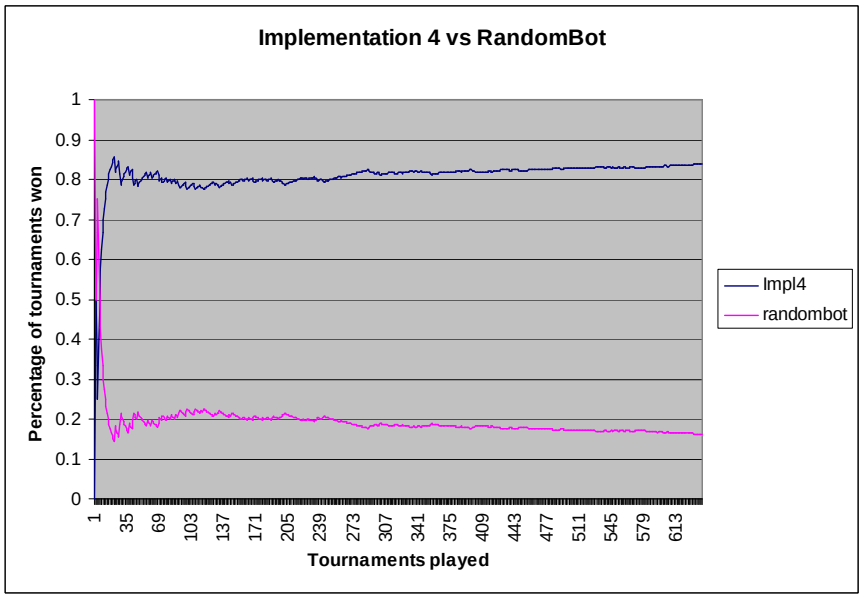


Figure 7.4.4

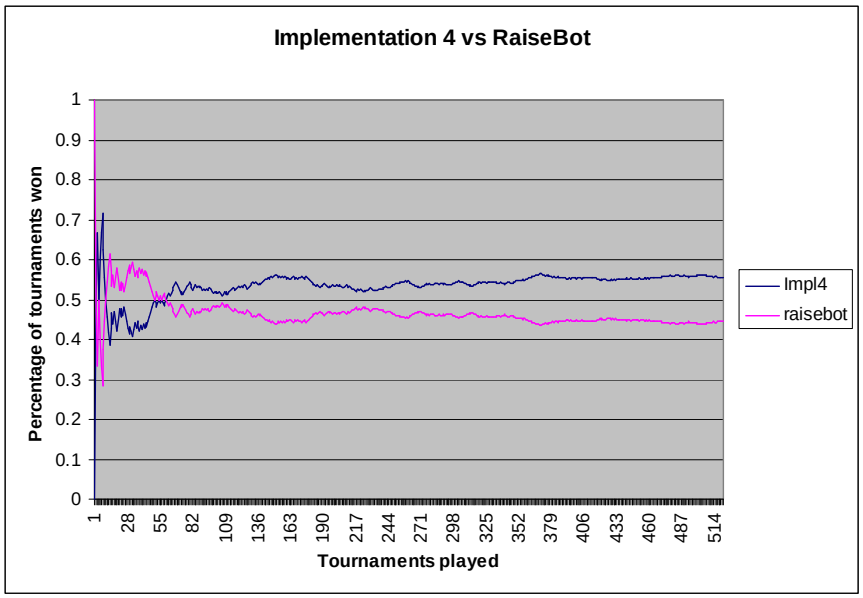


Figure 7.4.5

7.9.4 ANALYSIS OF RESULTS

Again, Figures 7.4.3 and 7.4.4 show that the underlying hand evaluation in the bot is sound. Figure 7.4.3 also shows that Implementation 4 attempts to find out what type of player its opponent is, and learns quickly how to play against it. Figure 7.4.5 shows the bot learning against an aggressive player, and the results show approximately the same success as Implementation 3 had against the AlwaysRaiseBot.

7.10.4 OVERALL

1) The bot can reason under uncertainty, with any hand that has a winning probability greater than the opponents' estimated best probability being played. There is no assumption that the opponents don't change their style, since a rolling average of hand probabilities of the past 50 hands is taken for each type of action at each particular stage.
2) The bot deals very well with probabilistic outcomes, with rollout simulations being used. Implementation 4 is possibly better than Implementation 3 since it uses only rollout simulations, and the somewhat unproven format of grouping the hole cards preflop strategy discarded.
3) The decision making action of how much money to bet is handled by a very simple function when no-one else has bet, but when someone else has bet, the amount to raise depends entirely on the bots' estimate of the best hand that an opponent holds. This is also linked to the aggression factor that the bot is initialised with.
4) The bot understands different styles of play. There is specific action opponent modelling, and whenever an opponent makes an action, an estimate of his hand's probability of winning can be made from the rating held in memory, and, if the opponent has called or raised, from the amount that he has called or raised. This is not limited to an

estimate of an opponents' overall tactic, since opponent action modelling is broken into different stages.

5) The bot understands different inferences of intent from opponents' bets, with an initial assumption that a certain action gives an indication as to what type of hand the opponent currently holds. An additional assumption is made that an opponent will bet a higher amount with a hand that is more likely to win, and further that the relationship between how good the hand is and how much the player bets is linear. This might be a reasonable approximation, however, a human player will normally employ some type of tactic to attempt to win more money, such as check raising or sandbagging, and so it will not always be accurate.
6) The bot understands deception to the extent that it will play a much lower rank hand than usual against a very loose and aggressive player. The bot might have been fooled by a player that changes tactics, however, the *testing the water* function means that the bot will continue to sporadically collect information about opponents hole cards by calling in preset situations, even though the opponent modelling and pot odds might suggest that this is not a good action. In this way, a player that changes from an initially tight aggressive player to a loose aggressive player does not bluff the bot off of pots forever.
7) There is no pattern recognition in the bot, particularly in the area of reacting to the amount that the blind is, and how much money is left in the bankroll compared to this.
8) The bot does not factor the impact of the amount of bets that it makes. This bot, however, will offer some deception of its own. When using the *testing the water* function, hands that might not have been played in the usual tactics could be played. Within the betting structure, if no one has bet before the bot, the amount to call is always zero. This means that if the bot is first to bet, a hand that isn't good enough to raise on average will be checked. If someone else bets, however, if the bot determines that his or her average probability in that situation is less than the probability of its hand winning, then it will raise. This appears as a check raise, and is very difficult to react to, even if the tactics of the bot are known.

7.11.4 POSSIBLE REFINEMENTS

The 4th implementation still has some drawbacks, mainly the blind recognition. The raise function is still determined by "magic" numbers; however, testing has shown these to be reasonable values.

Section 8: Multiplayer bot tournament

8.1 SETUP AND RESULTS

In order to test the implementations further, it was decided that a multiplayer tournaments needed to be staged. In order to run a reasonable amount of tournaments, humans could not be used. This meant using the only other no-limit poker bots available, those built by the others doing the same project as this one this year. Each person entered their best static (no opponent modelling) and their best opponent modelling bot. Because this is a multiplayer tournament, instead of simply looking at percentage of tournament wins, a mock tournament structure was set up. This dictated that each bot "paid" £16.66 in order to enter each tournament (this was in order to get the total amount to equal £100 per tournament). The player in first place "received" £70, and the player in second place "received" £30. All other players received nothing. This was run over 175 tournaments.

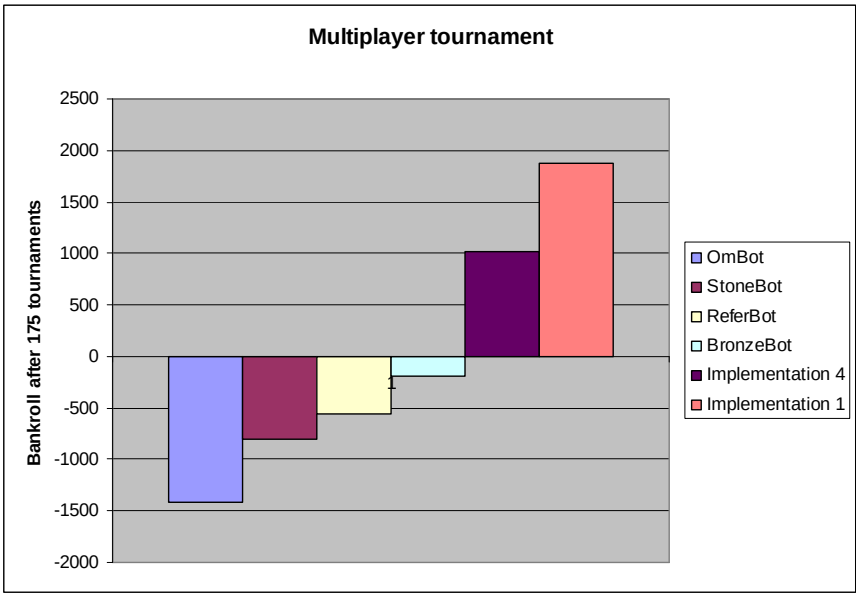


Figure 8.0.1

The results in Figure 8.0.1 show that Implementation 1 did the best out of all of the bots, with Implementation 4 coming in second place.

Section 9: Further testing: human opponent

The many tests performed against automated bots unfortunately show the first implementation as being the best. This, however, does not highlight the improvements that were made as the project progressed. Testing the final implementation against a human opponent best shows the difference made by adding opponent modelling to the implementation.

9.1 IMPLEMENTATION 1 VERSUS A HUMAN

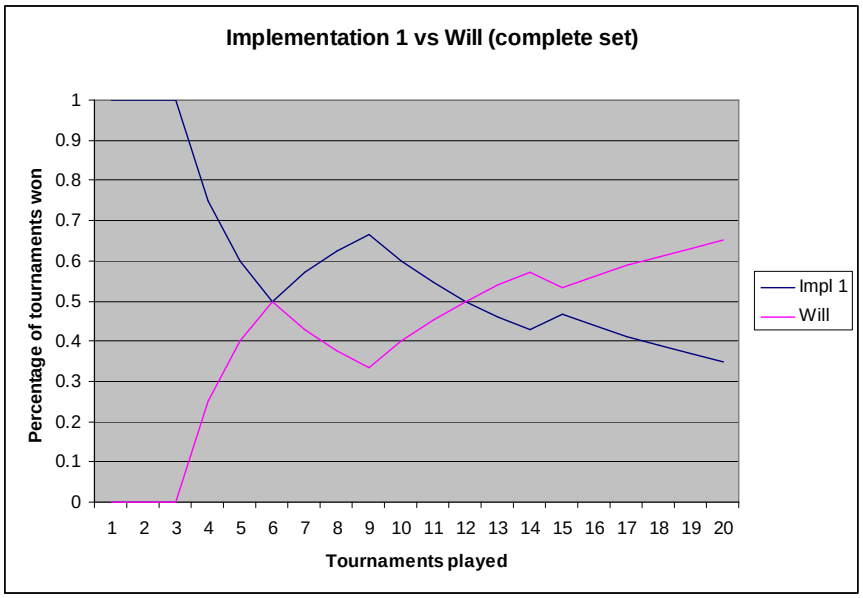


Figure 9.0.1

Figure 9.0.1 shows a series of 20 tournaments played between the first implementation and the supervisor of this project, a player with two years of no limit poker experience. Implementation 1 clearly makes a very good start, but Will eventually learns the bot's style of play, and ends up winning the majority of tournaments. Since this bot is static, it is fair to say that the style of its play has been learnt by this point, and that we would continue to see this bot beaten by a human player in any future situations. In order to further demonstrate the play observed, the results have been split into two, with Figure 9.0.2 showing the first ten tournaments, and Figure 9.0.3 showing the final ten tournaments.

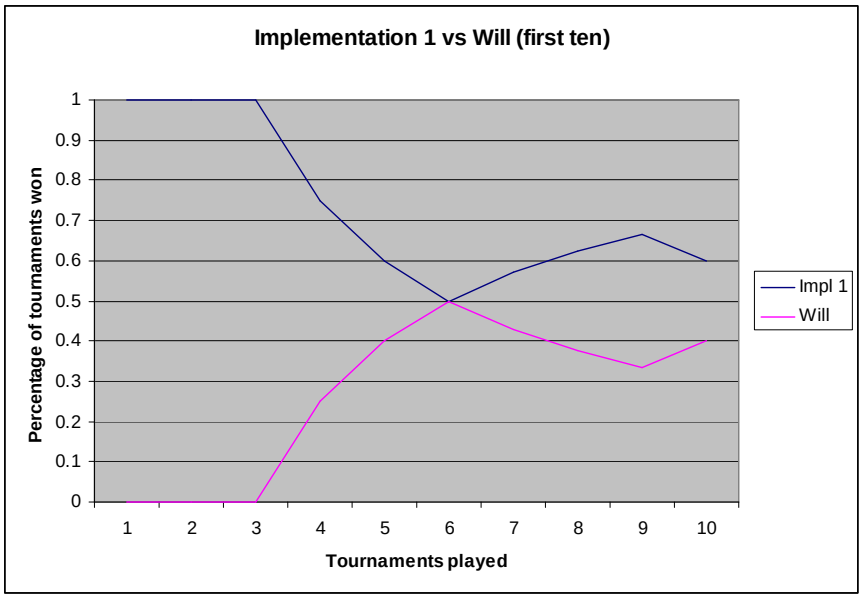


Figure 9.0.2

Figure 9.0.3

Figure 9.0.2 again shows the initial advantage that was held by the bot. Figure 9.0.3 highlights the advantage that was held by Will after he had learnt the bot's strategy. In fact, the bot only won one tournament out of the final ten.

9.2 IMPLEMENTATION 4 VERSUS A HUMAN

A similar set of tournaments was held between Implementation 4 and Will in order to provide a comparison. In this case, fifty tournaments were played, in order to thoroughly test the bot's opponent modelling.

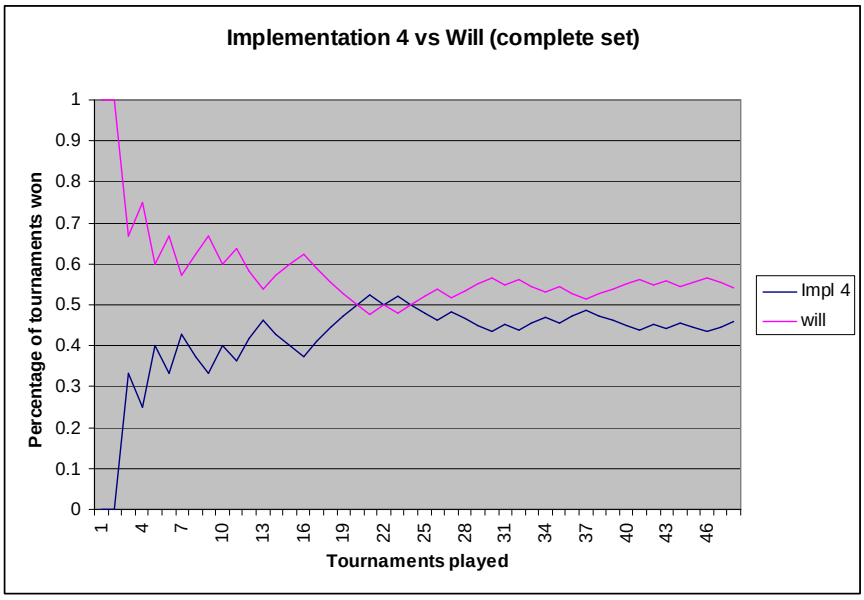


Figure 9.1.1

Figure 9.1.1 shows that although Will held an early advantage, eventually Implementation 4 learns the parameters within which Will plays, and starts winning more tournaments. As the bot is learning about Will's style of play, Will too is learning about the bot's style of play. In order to demonstrate how well Implementation 4 adapts, the results are again split, with Figure 9.1.2 showing the first ten tournaments, and Figure 9.1.3 showing the final ten tournaments.

Figure 9.1.2

Figure 9.1.3

Figure 9.1.2 shows the initial advantage held by Will, and the fight back by the bot after it learns. Figure 9.1.3 shows that Will has the measure of Implementation 4 in the final ten tournaments, however, as the bot continually adapts to its opponents play, its strategy cannot be said to have been learnt by Will. This is evidenced by the fact that the winning ratio of the final ten tournaments is the same as the first ten, in direct contrast to the tests run between Will and Implementation 1. As Figure 9.1.1 attests, the overall winning percentage held by Will is less than the percentage held in the first and the last ten tournaments.

In this sense, it can be said that the opponent modelling approach has had some success, even if it is somewhat limited.

Section 10: Conclusion and Discussion

10.1 COMMENT

The project progressed much as intended, with the goal of implementing a bot that achieved the cognitive reasoning in areas outlined by the ICCM. There was a refinement with every new bot created, and although these may have not been borne out in the overall results, the cognitive recognition was improved with every implementation.

Building a no limit poker agent and all of the related challenges represented an essentially un-researched field, with all background research done based on human strategy, machine learning, and limit poker agents. Although this research provided a starting point, particularly on the University of Alberta poker research group's papers, any implementations of the bots were original ideas and material.

10.2 STRENGTHS / WEAKNESSES

The project highlighted some sound tactics, and also some problems that any future project might encounter.

The first of the strengths to be pulled from the project is the hand evaluation. The 1st implementation proved that the function and idea behind the evaluation was sound. This stood the rest of the implementations in good stead, and did not need to be altered.

The second of the strengths was the performance of the 4th implementation against a human player heads up. Although it was predicted to do well when there were many human opponents, the strategy ensured that it was difficult to beat in heads up, and could surprise its opponent in many situations.

The third strength was the adaptiveness of each of the opponent modelling bots. Each has shown that they are capable of learning strategies of opponents, and reacting accordingly. Although some of the reactive strategies have been shown to be slightly limited, there is scope for extension of the rules in order to best utilise the statistics gathered.

Finally, coupled with the strengths mentioned above, is the fact that both Implementation 1 and Implementation 4 beat all other known no limit bots that have been created.

There were also several weaknesses to the project, which hampered many of the ideas and their implementation.

The main weakness encountered was the difficulty in evaluating the bots' play. The aim of the project was to build a good no limit poker bot. The structure of human championship no limit poker tournaments is to start with ten players at a table, then play the winners from each table at a new table, until one is left standing. Since the protocol designed was original to the project, any testing had to be done within the DOC environment. This lead to a lack of players, which meant that the tests were

performed against a standard set of players, and did not allow for very comprehensive testing.

Not only was there a lack of variety in the players, the way that the other players react with each other can majorly affect the way a tournament plays out. This meant that a truly scientific approach could not be taken, since the exact situation could not be set up every time and tested with different parameters for the bots.

These two facts restricted most of the evaluation to heads up tournament play, which is useful in some ways, but by no means a comprehensive test of a poker bot. This in turn meant that any improvements made to the bots were done simply in reaction to how well they did in these heads up tournaments.

As mentioned earlier in this report, the trends observed about a bot's play could be used in order to refine and improve the model. These observations, normally made within heads up tournaments, can normally extend to tournaments that contain more players. Evidence of this is borne out in the multiplayer tournament contained in section 8, where the same logic and strategy was employed, and the results were still favourable.

10.3 SUGGESTED EXTENSIONS

There are many extensions available to this project, the first of which must be the entry into next year's no limit poker bot world series [1]. It would be interesting to see how the bots in this project, particularly Implementation 1 and Implementation 4, would fare against the rest of the world's best.

For any project that wishes to build on any observations contained in this project, it would have to be suggested that the way in which a bot was evaluated would have to be thought about very carefully. Good evaluation would allow for better analysis of a bot's weaknesses and strengths, and therefore more sound data upon which to refine models would be available.

From the behaviour observed in some of the bots from this project, an interesting extension could centre on making a bot that plays as the first implementation, but plays as an AlwaysRaiseBot at random intervals. This, of course, would need no opponent modelling, but would be very difficult to read. If this could be coupled with a better evaluation method, then the results could provide a very good comparison to the worth of the opponent modelling in this project.

A further suggestion could be the development of a bot with the advice of an expert poker player. This approach, of course, would require the assistance of an expert player, but the advice could undoubtedly refine the model to result in a better implementation.

Section 11 Bibliography

[1] http://simon.lrdc.pitt.edu/~iccm/pokerbot.html

[2] http://boardgames.about.com/cs/poker/a/texas_rules.htm

[3] http://www.all-poker-rules.com/ranks-of-hands.html

[4] www.flopturnriver.com/start_glossary.html

[5] http://www.flopturnriver.com/poker-essay-contest-heatman.html

[6] Theory of poker - David Sklansky

[7] http://www.flopturnriver.com/essays_preflop_groups_0_to_2.html

[8] http://www-2.cs.cmu.edu/People/mummert/poker/

[9] [BAR99] Barone, L. and While, L. An Adaptive Learning Model for Simplified Poker Using Evolutionary Algorithms. In proceedings of the Congress of Evolutionary Computation (GECCO-1999), p. 153-160, 1999.

[10] http://www.doc.ic.ac.uk/~sgc/teaching/v231/lecture10.html

[11] Computer Chess MSc Individual Project Report September 2004 - Riccardo Ussani

[12] Machine Learning - Tom M Mitchell

[13] Fredrik A. Dahl: A Reinforcement Learning Algorithm Applied to Simplified Two-Player Texas Hold'em Poker. ECML 2001: 85-96

[14] Towards an Optimal Poker Playing Robot MEng Individual Project Report June 2004 Chun Leung

[15] www.cs.ualberta.ca/~games/poker/