Introduction

Abstract
Poker provides an interesting test bed for Artificial Intelligence research. Poker introduces factors and properties which are not present in other games such as chess or checkers. It introduces uncertainty through:

- Randomness in the form of shuffling of cards
- Imperfect information through not knowing the opponent's cards
- Limited knowledge of the opponent and what strategies they employ when playing.

In Heads Up poker, two players against each other. In general, sitting still and waiting for premium hands to come up as the game goes on is not a good strategy. This introduces another challenge for players.
This project attempts to create an Agent capable of playing heads up Texas Hold 'Em no limit poker. It also attempts to provide a user interface so users can play against it.

Project Objectives

This project aims to provide an means to play Heads Up no limit Texas Hold 'Em Poker against Artificial Intelligence.
It aims to:

- Provide a user interface using HTML5, CSS and JavaScript so that users can play poker using only their browser.
- Provide an interesting opponent that is relatively challenging to play against.

The AI opponent should take into account the following in order to create an interesting opponent to play against:

- Hand strength
- User playing style
- Possibility of deception (bluffing or sandbagging)

Design

The AI system design can be broken down into two parts:

- Postflop - decision network
- Preflop - rules based system

Rule based system - preflop
Why use a rule based system for preflop stage?

The idea behind using rules based system is to ensure that for the majority of hands played, that the AI agent will at least go the flop stage of the hand.

Rules based system design
The Rules based system takes inputs as follows:

- Hole cards
- Previous Action

Outputs:

- Action

Design for Decision Network - postflop
At a basic level, the system takes in a set of inputs and outputs an action, described as follows:

The inputs that the system takes in:

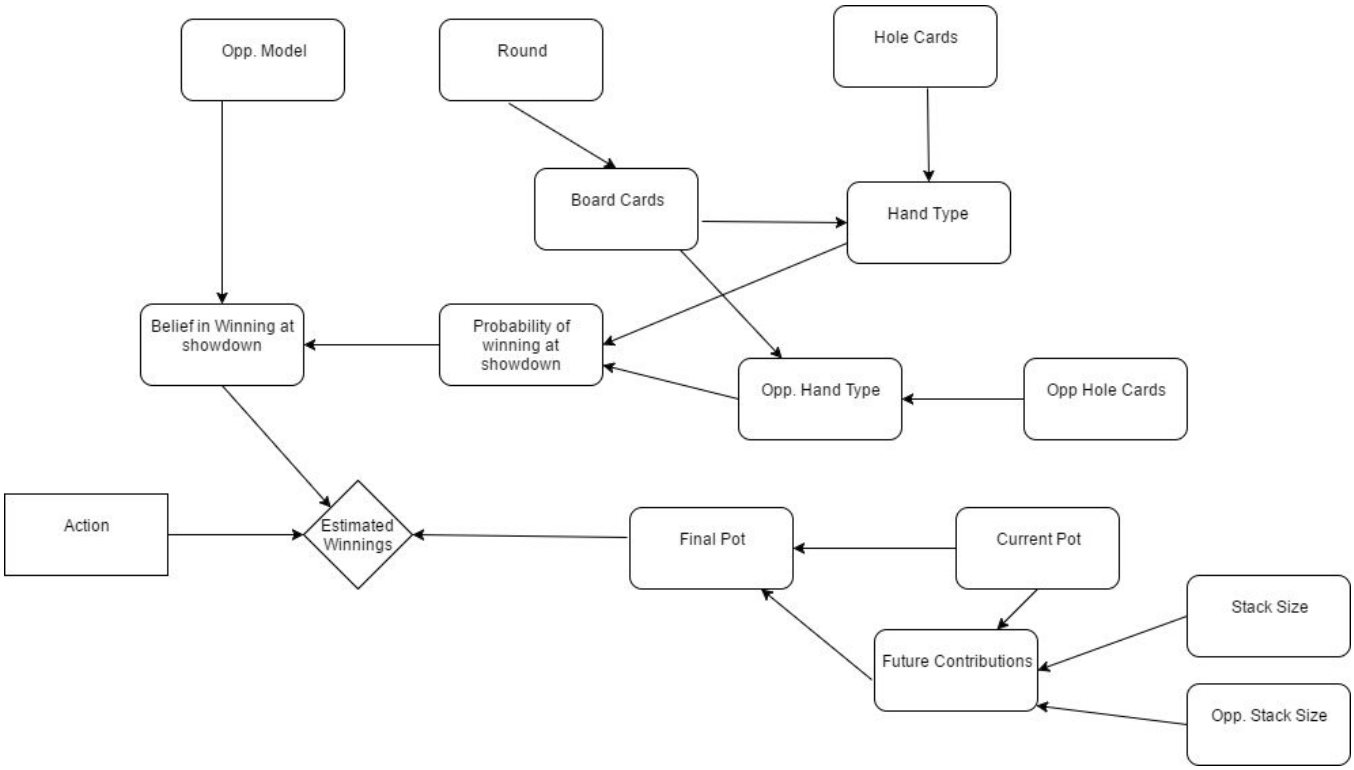
- Hole Cards
- Round
- Board Cards
- Stack Size
- Opponent Stack Size
- Current Pot Size
- Previous Action
- Amount bet (in previous action)
- The Opponent Model

The actions the system outputs (one of the following):

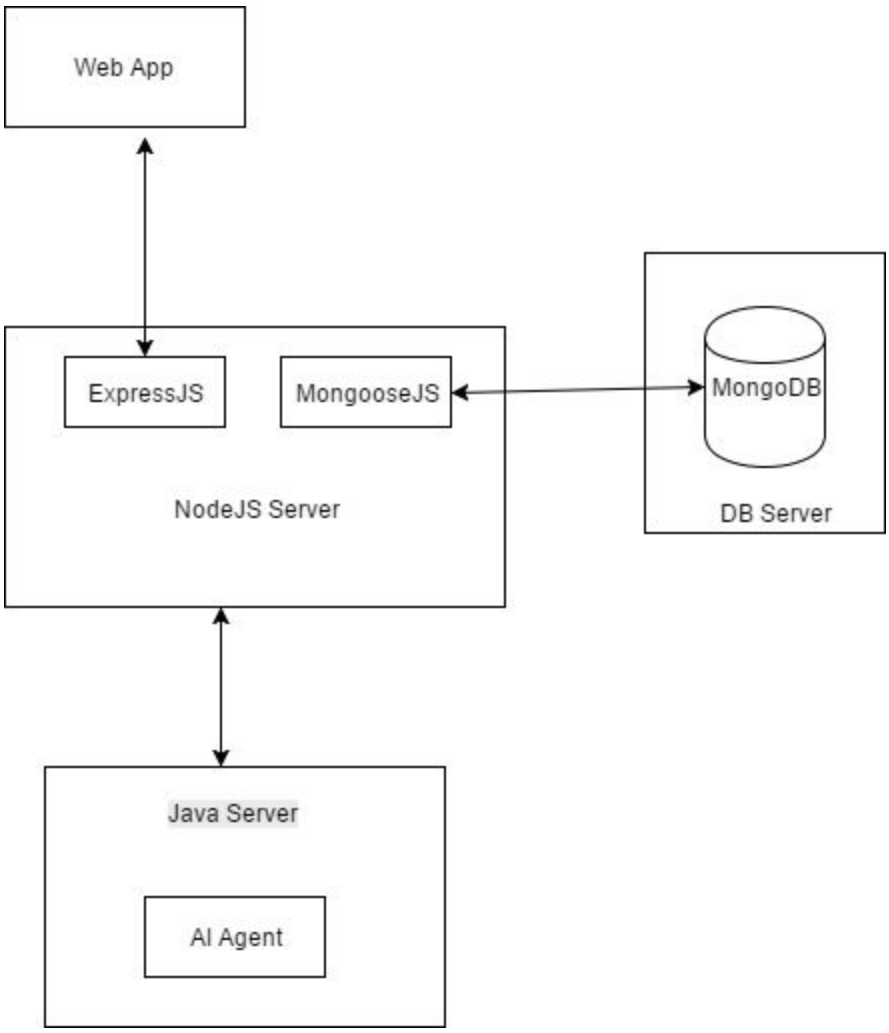
- Fold
- Check
- Call
- Bet1/Raise1 - representing betting ¾ of the current pot size.
- Bet2/Raise2 - representing betting (2 * ¾) of the current pot size.
- Bet3/Raise3 - representing betting (3 * ¾) of the current pot size.
- All In - Representing betting all of the current stack size.

The system takes the set of inputs, and uses them to produce a belief in winning the hand at showdown. A utility node estimates the winnings of the current hand given this belief, depending on what action the AI performs. The system then outputs the action, depending on the value determined from the utility node.

Diagram representing post flop network structure.



System Architecture Design



Implementation

Implementation of Preflop Rules based system

Key Components

Equity value

Each pair of hole cards has an associated 'Equity value' (EV) associated with them - how often they tend to win. A list of poker card rankings with their EVs (the EV is the average number of big blinds won per hand) associated with them can be obtained here (https://www.tightpoker.com/poker_hands.html). This data has been obtained from over 115 million pairs of hole cards dealt out at real money tables. As part of this project, the lists of cards has been mined from the page, with the help of the JSoup library.

Unpredictability

Unpredictability is an important part of poker - if an opponent's action can be predicted, they can be exploited. Therefore it is important to have an element of randomness as part of the implementation.

Pseudo Code Implementation

```
Inputs: holecard1, holecard2

Int foldThreshold = 90
Int rankThreshold = 40
Int checkCallThreshold = 66
Int Bet1Threshold = 75

Int evRank = getEvRank(holecard1, holeCard2)
Int randomnum = getRandom()

If (evRank > rankThreshold && randomNum > foldThreshold) {
       Action = fold
}
If (randomNum < checkCallThreshold) {
       Action = checkCall
}
If (randomNum < bet1Threshold) {
       Action = bet2
}
else {
       Action = bet1
}
Return action
```

Drawbacks

This implementation needs formal validation through self play to validate the best parameters for this. However, preliminary user testing has shown that this implementation does what it is intended to:

- Goes into the next round of betting most of the time
- Is not entirely predictable in terms of its action output

Implementation of Decision network

Key components

Estimated Winnings

The estimated winnings represents the value of the number of chips that the system estimates it will win if the hand goes to show down. The system uses the methods described in Carltons paper to work out the final pot and future contributions to the pot.
The way in which the future contributions are calculated depends on what the proposed output action is:

- If the action is Bet
    - Future contribution = amountToCall + betSize + (roundsLeft * betSize) * bettingFactor
- If the action is to call or check
    - Future contribution = amountToCall + (roundsLeft * betSize) * checkCallFactor

The opponent's contribution is calculated in a similar manner:

- If the action is Bet
    - Opp. future contribution = betSize + (roundsLeft * betSize) * bettingFactor
- If the action is to call or check
    - Opp. future contribution = (roundsLeft * betSize) * checkCallFactor

The final pot is calculated as follows:

- Final Pot = future contribution + opp. Future contribution + current pot

The betFactor and checkCall Factor constants were obtained from Carltons research - they were determined stochastically using a greedy algorithm. Validation in the form of self play would be able to show whether these values produce optimal results for this system also.

Utility Table
The utility table was also obtained from Carlton's work. It is defined as follows

Action	**Outcome (Win/ Lose)**	**Utility**
Bet	Win	Final Pot - future contribution

Bet	Lose	- Future contribution
Check/Call	Win	Final Pot - future contribution
Check/Call	Lose	- Future contribution
Fold	Win	0
Fold Lose	Lose	0

Note: Folding always leads to a utility of 0 as regardless of the probability of winning at showdown, there is no further chance of winning or losing in the current hand

The system uses the belief and each action/outcome pair to calculate the estimated winnings for each betting action specified in the table.

Belief in Winning at Showdown

The belief in winning at showdown is largely influenced by a combination of the probability of winning at showdown and also the opponent model. For example, a higher probability of winning at showdown would lead to a higher belief of winning at showdown. However the doubt of winning at showdown inferred from the opponent model affects the belief also.
In pseudo code this can be represented as follows:

```
belief = ProbabilityOfWinningAtShowdown
for ( round IN roundsOccuredSoFar ) {
        belief = belief-opponentModel.getFoldRatio(round)*roundConstant
}
```

The exact values of the round constants need to be validated and documented in the search optimum results.

Opponent Model

If no opponent model is inputted to the system, it will use the 'default' opponent model obtained from no limit Texas Hold 'Em playing data supplied by the University of Alberta.

An opponent model has the following attributes associated with:

- Ratio of folding at pre flop
- Ratio of folding at flop
- Ratio of folding at turn
- Ratio of folding at river

The idea behind choosing these specific attributes is so that the network can produce an accurate idea of 'doubt' to winning the hand showdown. These attributes were chosen so that the system would be able to adapt to the playing styles of both tight and loose styles of players. The higher value these attributes have, the more doubt is created - this doubt is then taken away from the belief of winning at showdown.

Results have been obtained, using self play and different opponent models, have been documented in Appendix D. The default opponent model won against all other models apart from the 'tight' opponent model.

Current drawbacks
The opponent model does not have any other betting patterns implemented as part of it. More experimentation is needed . Ideas include having betting patterns influence the belief, aggressiveness vs passiveness.

Hand Type
A hand type for the AI is determined by using the inputs of the board cards and hole cards. The hand type is then classified into a 'bucket', which generalises the types of hands that come up with poker. The bucket table currently has been implemented with twenty five entries (described below). Each entry has a probability of occurring associated with it. This has been implemented in the HandType class of the AI_Agent module.

Table obtained from Carlton's paper (Table 2)

Hand Type (Bucket)	**Probability**
Busted Low (Busted 9 or lower)	0.0203227
Busted Med (Busted 10 or Jack)	0.07631
Busted Queen	0.082212
Busted King	0.1292463
Busted Ace	0.193495
Pair of Twos	0.0325
Pair of Threes	0.0325
Pair of Fours	0.0325

Pair of Fives	0.0325
Pair of Sixes	0.0325
Pair of Sevens	0.0325
Pair of Eights	0.0325
Pair of Nines	0.0325
Pair of Tens	0.0325
Pair of Jacks	0.0325
Pair of Queens	0.0325
Pair of Kings	0.0325
Pair of Aces	0.0325
Two Pair	0.0474187
Three of a kind	0.0211247
Straight	0.0038647
Flush	0.0020007
Full House	0.0014023
Four of a kind	0.0002347
Flush Straight	0.000014

Why twenty five buckets?

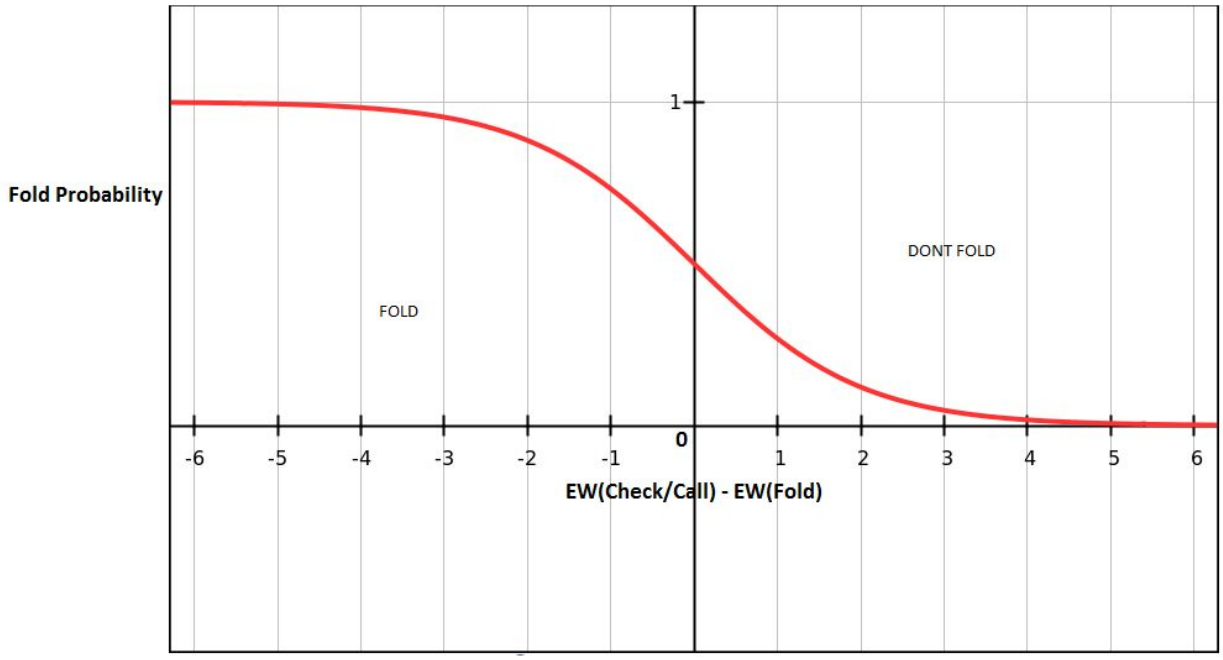
The reason for choosing twenty five buckets is because of evidence shown in Nicholson et al. work showing that this implementation with these entries provide sufficient enough granularity in order, and an increase in the number of buckets do not necessarily represent a great gain in terms of winnings. However, further testing in the form of self play must be done in order to verify the effects of changing the number of buckets in the table in order to find the optimal amount of buckets for this system.

Action/Betting Curves

From the estimated winnings, an optimal action (fold, check/call, bet) can be determined - by picking the the action with the highest estimated winnings. However, if this deterministic strategy was purely used then the system may become predictable against opponents, who may quickly pick up that the system carries out an action purely based on what it believes is correct action to carry out - it would not attempt to try and deceive opponents by trying to bluff or sandbag in its gameplay. In order to address this issue and using the work of Carlton and also Nicholson et al., a solution by way of using betting curves and randomisation is implemented as part of the system. The system uses three different curves to determine what action to carry out:

Fold/(Check/Call) Function (Carlton)

Fold probability = 1 / (1 + e ^ (foldRoundParam * (EW (Check/Call) - EW (Fold))))
Input: EW(Check/Call) and EW(Fold)
Output: Fold probability
Graph of fold probability function:

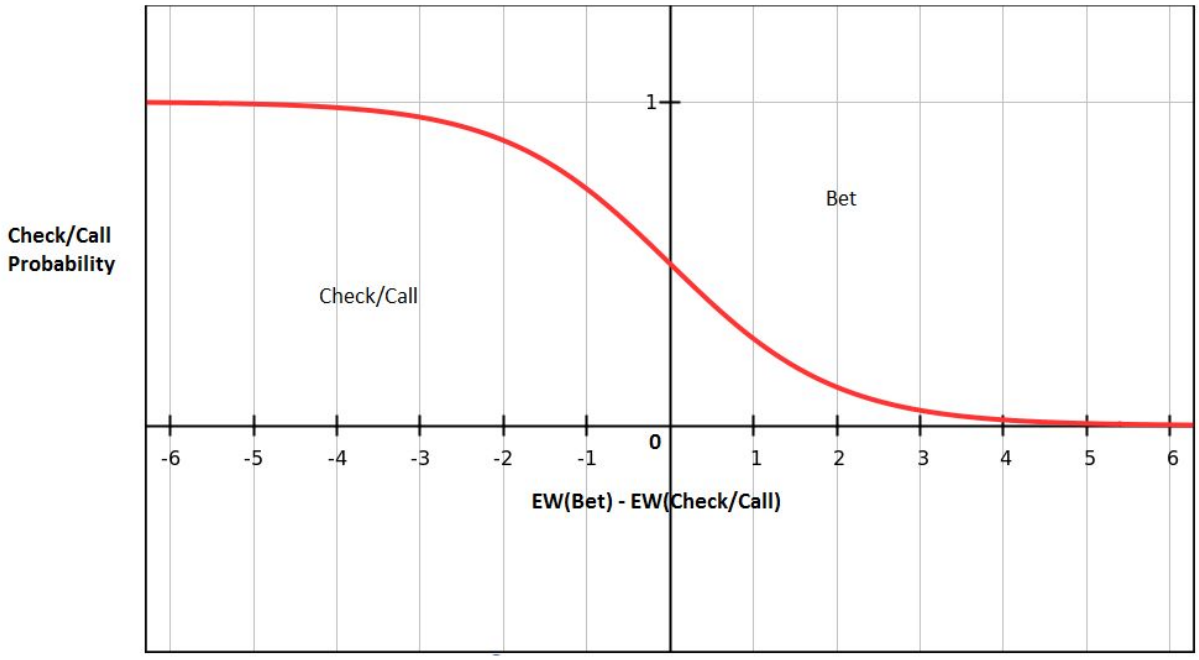


In order to determine an action, a random number is generated. If the random number is less than the fold probability calculated, than a fold action is outputted.
If it is greater than or equal to the fold probability, than the system moves on to determine whether it should just check/call or bet using the check/call / betting function.

Check/Call / Bet Function (Carlton)

Check/Call Probability = 1 / (1 + e ^ (checkCallRoundParam) * (EW(Bet) - EW(Check/Call))))
Input: EW(Bet), EW(Check/Call)
Output: Check/Call Probability

Graph of Check/Call Probability function

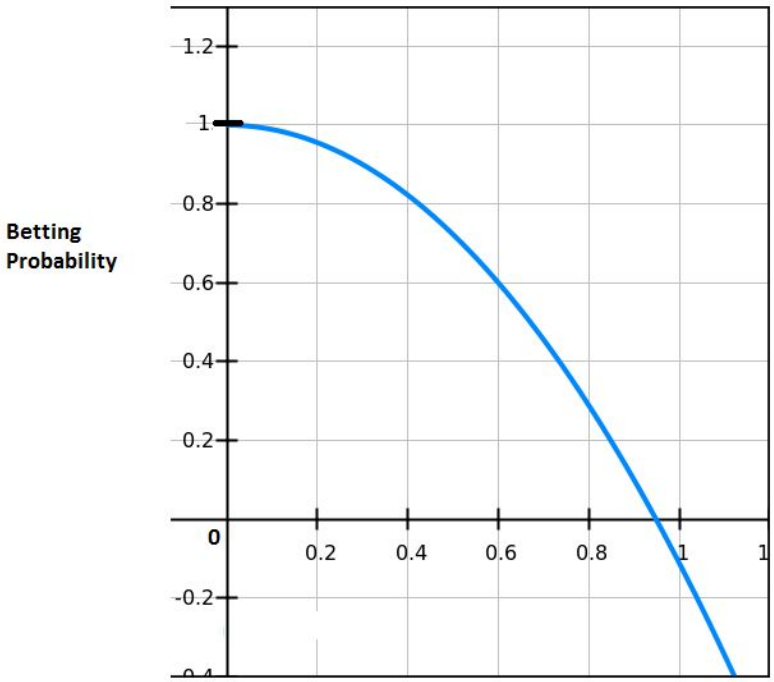


The process of determining the action is much the same as the fold/(check/call) function.
A random number is generated. If the random number is less than the check/call probability calculated, than a check/call action is outputted.
If it is greater than or equal to the fold probability, than the system moves on to determine what type of bet it should output.

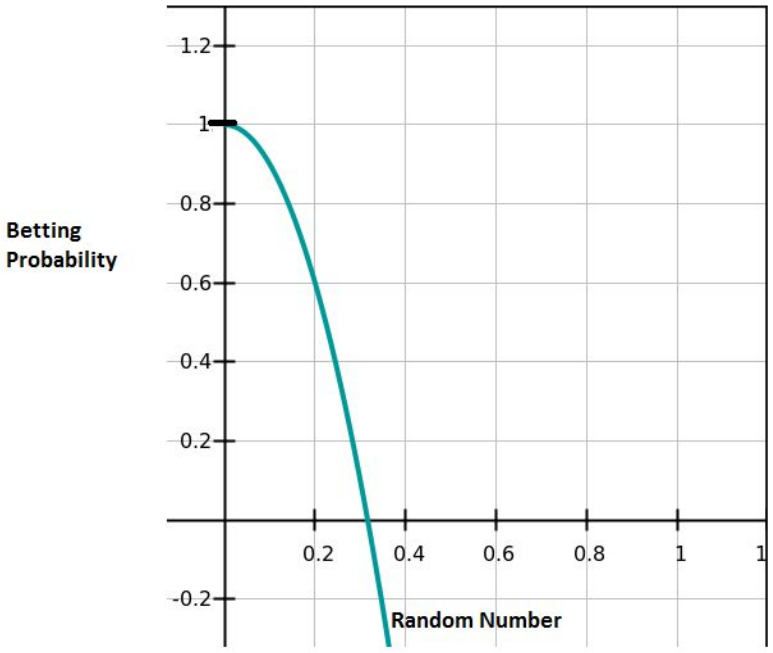
Note: The foldRoundParam and checkCallRoundParam vary from round of play (flop, turn, river) were obtained from Carltons' research. They were determined by using a stochastic function searching for the optimal values. Validation in the form of self play needs to be done in order to observe whether these are the optimal values for this system.

Betting Function

Inputs: p = Check/Call Probability and x = random number.
Output: Betting Probability
Graphs given sample values for pass probability:



Where Pass Probability = 0.1



Where Pass Probability = 0.9

Depending on what the betting probability is, an action will be outputted. These values have been determined through documented experimentation through self play. This can be found in appendix B. The following actions can be outputted from this:

- Bet1
- Bet2
- Bet3

- All In
- Check/Call

The lower the value of pass probability, the more the positive y values stretch over the x axis. This makes it more likely that an optimal action is chosen, as per the pass probability, however also allows room for a suboptimal action to be carried out. This can be observed and interpreted as bluffing (deliberately over valuing cards) or sandbagging (deliberately undervaluing cards), and this deception is a vital part of poker and not becoming predictable in carrying out actions. This has been observed as part of the user testing carried out - this can be found in appendix A.

The implementation of these functions can be found in the ActionDeterminer class of the main AI_Agent module of the project.

Opponent Modelling - forming a default opponent

Opponent Modelling plays an important part of computer poker and developing this kind of system. It plays a part in attempting to determine what style of poker playing that an opponent plays as. Players playing styles could be described as the following:

- Tight: More likely to fold in the early rounds (preflop and flop) if they do not have a good hand.
- Loose: Less likely to fold in the early rounds (preflop and flop). Are willing to play most hands regardless of hand strength.

Opponents can also be classified in terms of their betting strategy:

- Aggressive: Players who tend to bet and raise frequently.
- Passive: Players who tend to check or call more frequently rather bet or raise. Extremely passive players are sometimes 'calling stations' in professional poker circles.

As part of forming the default opponent model used for this project, both these traits were taken into account. Data obtained from the University of Alberta Poker group was used for this. The data used described No Limit Texas Hold 'Em hands and players actions. Information on how this data was recorded can be found here: http://webdocs.cs.ualberta.ca/~games/poker/irc_poker_database.html.

Only heads up games were used as part of the data analysis carried out in order to form a default opponent model.

How was the default opponent model formed:
The default opponent model was formed by implementing a clustering algorithm, with the idea being to form different clusters of players, representing different types of players who have different playing styles.

A hierarchical hierarchical agglomerative clustering algorithm. It was preferred to the other main method of forming clusters, k-means clustering, was that the points chosen in k-means to be the

'centroid' points affect the output of the clustering drastically. The main drawback to the algorithm used was the fact that it runs in order n^2 time - however the time taken to form clusters was not too much of a concern, so long as it was not overly unreasonable.

The algorithm was implemented as follows:

1. Assigning each point (a player) its own cluster
2. Computing the distance between each cluster
3. Merging the two closest cluster together forming their own cluster
4. Using the smallest distance between the other clusters and the most recently merged cluster to compute the new distances
5. Repeat steps 2-4 until a certain number of clusters determined.
6. Calculate the centroid of each cluster.

This parsing of the data implemented in the Data_Processing module of the project.
The clustering of data is implemented in the dbprocessor package of the main AI_Agent module.

The results of the clustering were used as follows:

- The cluster with the greatest amount of players is used as the default opponent model.
- The other clusters formed were used as part of validation and evaluation of the AI system in testing it against different opponent models.

Flush / Straight Prediction - partial straight and flush identification

As part of Nicholson et al. work, it was found that flush and straight prediction through partial straight and flush identification could help increase winnings. Straights and flushes are amongs the highest handing hand types.

Partial straight and flush identification is implemented as part of the StraightFlushChecker class. If partial straights or flushes were identified, then the value of belief in winning at showdown is updated to reflect this.

Promising results have been shown through self play. A system with the ability to identify partial straights and flushes won an average of 0.286 +- 0.1618 big blinds per hand compared to a system where partial identification of flushes and straights did not influence belief. Documented results can be found in Appendix C.

Common hand

Noted in observing the system's play, it seemed to be playing extremely extremely aggressively when a pair or high card appeared in the board cards. It was felt that this was possibly an area where a player could notice this the system could be exploited. To counteract this, a CommonHand class was implemented as part of the system. The CommonHand class updates the belief in winning at showdown

if it identifies that the best hand type is contained solely in the board cards. If this is the case, the belief in winning is reduced.
Through self play, it was shown that the system with this belief updating performed worse than the system without this belief updating. The system without this won on average 0.2657 +- 0.104 big blinds per hand more than the system with this implemente. Documented results can be found in Appendix C. While these results indicate that this belief updating is not a worthwhile addition to the system, it remains to be seen if this trait can be in fact be exploited by users or not.

System Architecture Implementation

Web App

The Web App makes use of the following third party libraries:

- AngularJS - https://docs.angularjs.org/api
- BootStrap - http://getbootstrap.com/
- Angular UI Bootstrap - https://angular-ui.github.io/bootstrap/
- CSS Playing Cards (Selfthinker) - https://github.com/selfthinker/CSS-Playing-Cards
- SocketIO - https://socket.io/

Design

The web app is a HTML5 AngularJS app that users can open and use with their browsers. It is made up of the following components:
Hierarchy:

- Table
    - Player
        - Chip Stack
        - Cards
        - Dealer Button
    - AIPlayer
        - Ai Action Message
        - Chip Stack
        - Cards
        - Dealer Button
    - Pot
    - Community Cards
    - Min-Bet

The Table Controller controls the game logic and flow. It also sends socketIO messages to the NodesJS server, and listens in for socketIO messages. It controls the appearance of its children components through use of attributes that can be defined using the AngularJS framework. Its implementation is largely event driven, either waiting on user input (e.g. clicking on bet, fold, or check) and reacting to that as appropriate, or waiting on socketIO from the NodeJS server (e.g. an AI action) input and reacting to that as appropriate.

There is also a login component that controls how a user logs in. At the moment, there is no password field or register an account option, a user can simply log in using a username. If the username is not valid, a modal screen will appear informing them that the username is invalid.

NodeJS Server

The Node Server is in charge of communicating between the web app, accessing the database, and the Java AI Server.

Components

- Uses ExpressJS https://expressjs.com/, a web development framework used in order to serve files to clients.
- Uses SocketIO, https://socket.io/ - a real time bidirectional event based framework to communicate to the web app. It listens for socketIO communications coming into port 3000.
- Uses MongooseJS (http://mongoosejs.com/) to model the application data and also access the MongoDB instance.
- Uses sockets (node 'net' module) (https://nodejs.org/api/net.html) to communicate with the java server.

Client Tracking

The node server tracks clients (clients referring to the users using the web app) using two javascript lists, tracking the user names and the socketIO sockets being used. This is to ensure smooth communication as information is communicated from the web app to the NodeJS server, to the Java Server and back again.

Poker Hand Evaluator

The NodeJS server also makes use of the Poker Evaluator node module in order to determine the winner at showdown. It identifies which hand is the winning hand and identifies the type of the hand (e.g. one pair, high card, three of a kind etc.).

MongooseJS

How MongooseJS is implemented can be viewed in the db.js file in the server folder of the ui module of the project. It controls the storing of opponent models in MongoDB. The opponent models can then be retrieved using the username that a user logged in with. The opponent model schema in MongooseJS is defined as follows:

```
var playerSchema = mongoose.Schema({
                        name: String,
                        numHandsPlayed: Number,
                        numFoldsPreFlop: Number,
                        numFoldsFlop: Number,
                        numFoldsTurn: Number,
                        numFoldsRiver: Number,
                        numGamesPlayed: Number,
```

```
    numHandsWon : Number,
    numGamesWon: Number,
    totalFolds: Number
});
```

The db.js file also controls how many hands it takes before the actual opponent model of an opponent is used, rather than the default. This number is currently defined in the following variable:

```
var NUM_HANDS_BEFORE_NOT_DEFAULT = 25;
```

It is unknown if 25 is the correct threshold for this. It may be the case that a higher number of hands is required before switching from the default opponent model, in order to create a more accurate model. More testing and validation with users is required here.

NPM

The node package manager(NPM), was used to manage dependencies in the UI module of the project. All specific dependencies can be viewed in the package.json file.

GulpJS

GulpJS was a tool used to help development of the web app. Gulp was used to minify css, concatonate all javascript files into one and also provided automatic restarting of NodeJS server when required.

Java Server

The java server uses sockets (Socket and ServerSocket classes in Java library) to listen in from connections coming in on port 3500. Once a connection is established a new thread is spawned. Once the input stream from socket has been read, the thread creates an Ai_Agent object and feeds the inputs obtained from the input stream to it using the getAction() method of the AI_Agent object. This action is sent back to the NodeJS server using the DataOutputStream class in the Java Library.

Input to java server:

- Username
- AI Agent inputs (Previous action, amount bet, round, hole cards, stack size, pot size, opponent stack size, board cards, opponentModel)

Response from java server:

- Username
- Action (Fold, Check, Call, Bet1, Bet2, Bet3, Raise1, Raise2, Raise3, Allin)

Evaluation

See test plan in appendix A documenting the testing activities.

Conclusions and Future Work

This project provides fulfills the following objectives:

- Developing a system for a user to play heads up no limit Texas Hold 'Em poker against the computer
- Provides an interesting opponent to play against

The system provides an AI opponent which takes into account the following:

- Hand Strength
- Opponent Modelling (to an extent)
- Possibilities of bluffing or sandbagging (to an extent)

It has not yet determined at what exact level the system plays at (weak amateur, medium level amateur etc.) however it is clear that this project may provide a good baseline to future work on this project.

Future Work

User testing the AI play

More extensive user testing is needed to verify the overall level of the AI systems playing style and strategies. Ideally, extremely experienced pokers would be able to point areas of weakness or exploitation and ideas could be developed in order to counteract this. Although self play has proved useful as a relatively quick way to validate decisions and parameters, these do not translate to actual playing strategies against real opponents and users.

Improving Deception plays

At the moment, bluffing and sandbagging only occurs randomly, in accordance with the betting curves defined. However a method of bluffing more regulularly

Opponent Modelling

As part of mastering the game of poker, a player must be adaptive to the opponent they are playing. At the present moment, only folding actions are taken into account as part of the opponent model. Future work could include working in common betting patterns that an opponent would commonly undertake. Deviations from these normal betting patterns could influence belief in the system.
A suggestion could be to carry out further analysis on the University of Alberta data set, looking for common betting patterns, and perhaps associating these patterns with each cluster of players.
Another point of future work would be trying to find out the point in which enough data has been acquired about a user that a user's own model should be used, rather than the default.

Modifying betting parameters based on opponent models

It could be the case that certain parameters perform better against certain types of opponent models. More experimentation and validation in the form of self play and against real users is needed to

investigate whether the effects of changing certain parameters lead to better and more successful outcomes.

Preflop
The preflop system has been developed in such a way so that it ensures that the system will be able to reach the next stage (flop) of the hand. However, it remains to be seen if this system can not be easily exploited - more extensive user validation from more experienced poker players must be examined here.

Decoupling UI from socketIO
The UI is highly coupled to the socketIO. There is refactoring work required in order to carry this work out.

Unit Testing UI
There are presently no unit tests for the UI - this would probably go hand in hand with the refactoring work with the UI. Possible frameworks to help develop these unit tests could include KarmaJS and Jasmine.

UI passwords and security
Currently there are no passwords or security as part of the system - anyone can log in with any allowed user name. Future work could include creating a register account function, allowing users to create a specific account for them with a username and password.

Automated Integration Tests and Acceptance Tests
Presently there are no automated integration tests or acceptance tests:

- Integration Tests would require the writing of test drivers checking for the expecting result.
- Acceptance tests could be written using frameworks such as the Node module 'Protractor' which provides end-to-end testing functionality for Angular apps.

References

CPRG- University of Alberta Poker Group. University of Alberta - http://webdocs.cs.ualberta.ca/~games/poker/

Carlton, J. (2000). Bayesian Poker. Honors Thesis, Monash University.

Nicholson A.E., Korb K.B., Boulton D. (2006) Using Bayesian Networks to play Texas Hold 'Em Poker. http://www.csse.monash.edu.au/bai/poker/2006paper.pdf