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 allows users to play Heads-Up Texas Hold 'Em no limit poker. Heads-Up poker is one-on-one poker in which the blinds increase as the game progresses. The project consists of the Poker AI, developed in Java, and an AngularJS/HTML5 front end which allows users to play a game against the AI in their web browser.

Project Objectives

This project aims to provide a 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)

Table of contents:

AI Design

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

- Preflop - rules based system
- Postflop - decision network

Rule based system - Preflop

Why use a rule based system for preflop stage?
The idea behind using a 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

Implementation of Preflop Rules based system

Key Components

Equity value

Each pair of hole cards has an associated 'Equity value' (EV) associated to 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 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 parameters (randomNum and FoldThreshold) have been examined and validated through self play. This has been documented in Appendix E.

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

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 the system takes in are:

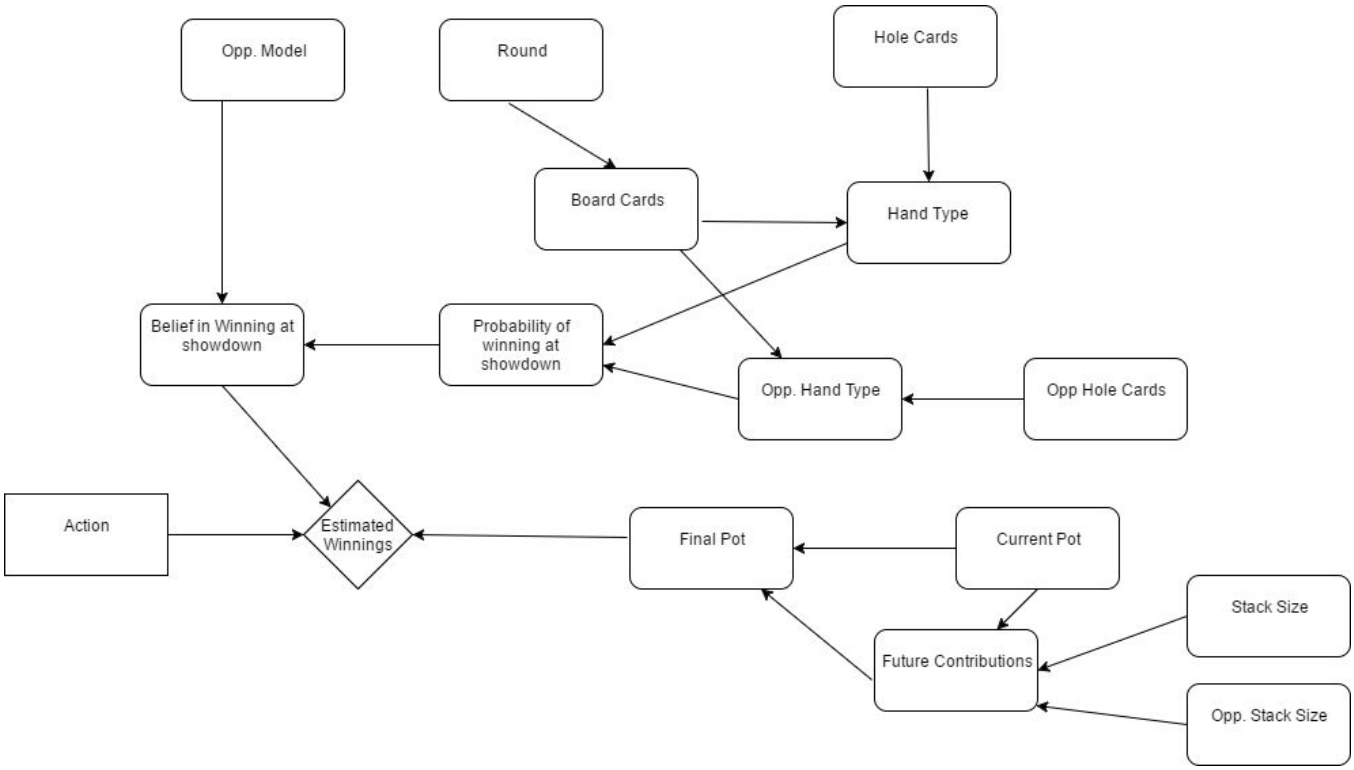
- Hole Cards
- Round
- Board Cards
- Stack Size
- Opponent Stack Size
- Current Pot Size
- Previous Action
- Amount bet (in previous action)
- The Opponent Model

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.



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.
Note: roundsLeft refers to the stages of play left - flop, turn, river.
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 to it:

- 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

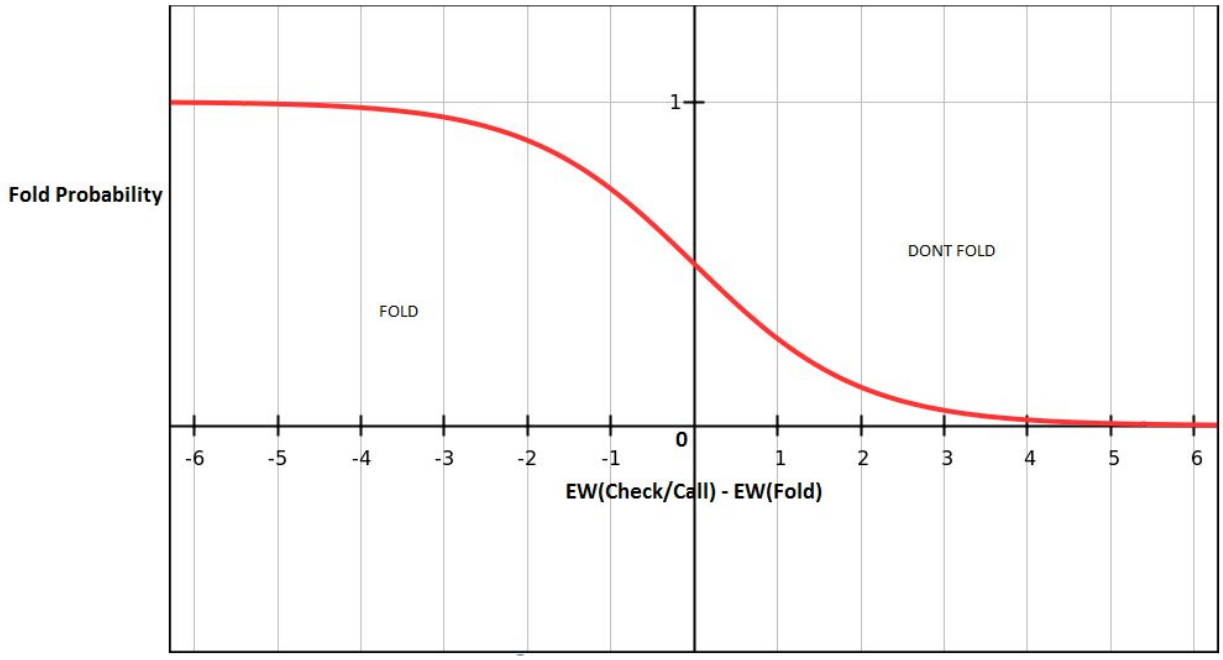
Three of a kind	0.0211247
Straight	0.0038647
Flush	0.0020007
Full House	0.0014023
Four of a kind	0.0002347
Flush Straight	0.000014

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 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:

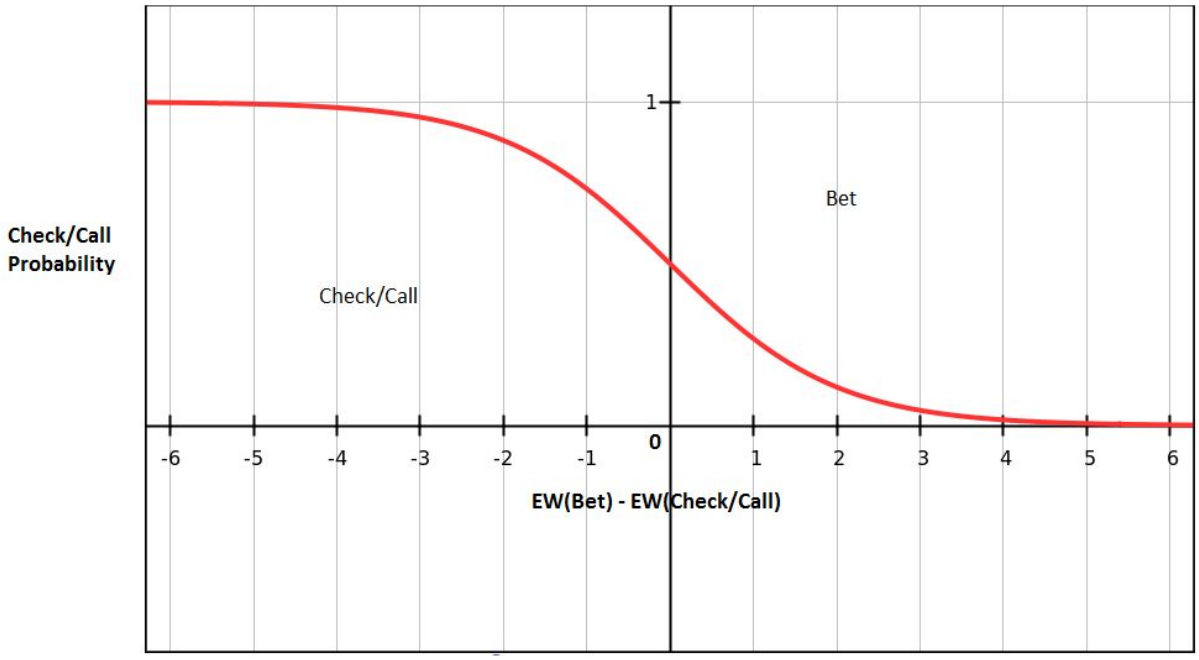
Fold/(Check/Call) Function (Carlton)

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

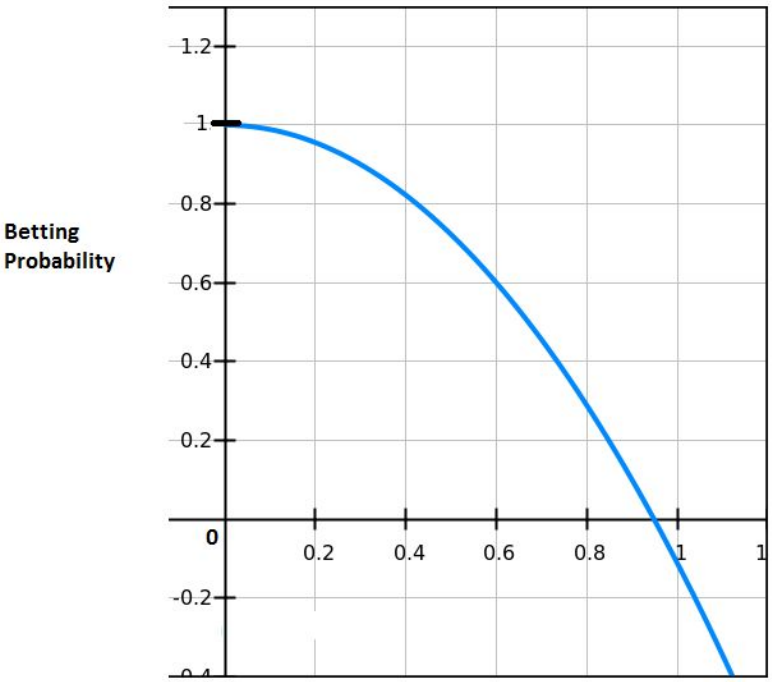


The process of determining the action is slightly different to the previous function. The function takes the EW inputs to obtain a check/call probability. This output is used as part of the next function.

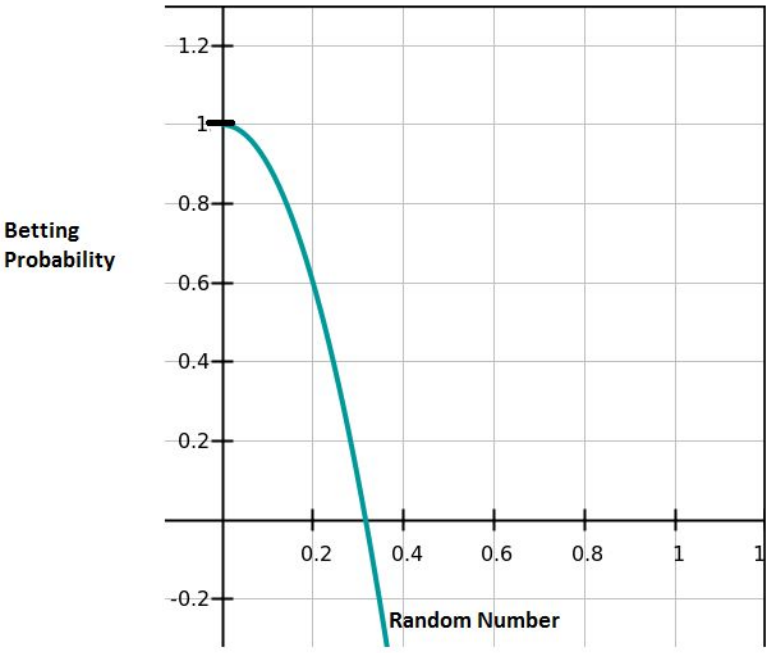
Note: The foldRoundParam and checkCallRoundParam vary from round of play (flop, turn, river) were obtained from Carltons' research. They were determined by using a stochastic function searching for the optimal values. Validation in the form of self play needs to be done in order to observe whether these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

Betting Function

Function: -((x^2) / (1 - p)) + 1
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, it 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 an opponent plays. 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), and 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 the default opponent model was 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 was used to form the default opponent model. The primary reason this method 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 amongst 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

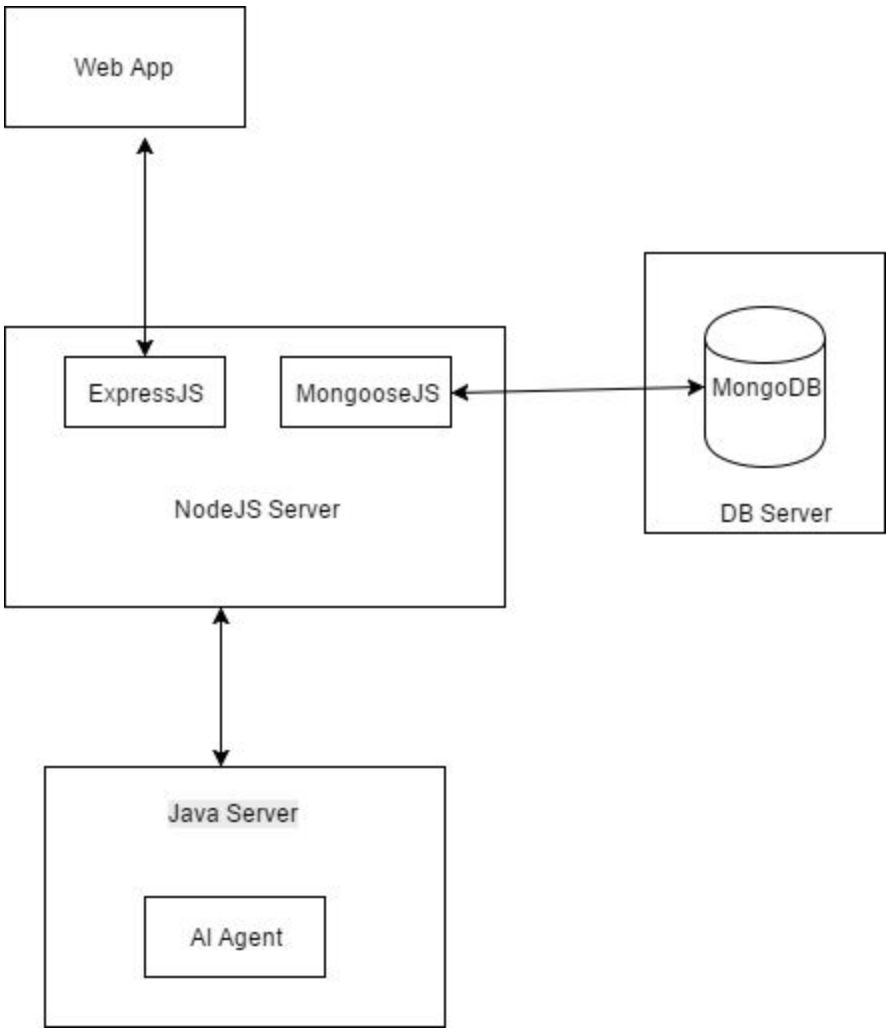
Noted in observing the system's play, it seemed to be playing extremely extremely aggressively when a pair or high card appeared in the board cards. It was felt that this was possibly an area where a player could this part of the system. To counteract this, a CommonHand class was implemented as part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

System Architecture Diagram



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, All-in)

Evaluation

See test plan in appendix A documenting the testing activities.

Conclusions and Future Work

This project provides fulfils 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 in a more structured way would be an interesting idea to implement. Potential ways of doing this would be artificially increase the belief in winning every certain amount of hands, or indeed to artificially lower the belief.

Opponent Modelling

As part of mastering the game of poker, a player must be adaptive to the opponent they are playing. At present, 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 flop. 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

Testing - Appendix A

Test Plan

The following testing activities have been carried out as part of this project

- Unit Testing
- Integration Testing
- Acceptance Testing
- User Evaluation Testing

Unit Testing

Automated unit testing should be completed on every module/component developed as part of the project, unless given reasons as to why not. Unit tests should be run with a tool that generates a report detailing code coverage numbers (e.g. class coverage, method coverage, line coverage) to be used as an indicator whether the unit tests are helpful or not (e.g. to help avoid having two tests doing the exact

same thing). Unit tests should test ideally test one specific thing or unit (e.g. method in a class) and use mock data where appropriate.
Where possible, the unit tests should have an error message detailing the reason as to why it would fail. These messages will serve as part of unit test documentation.

Integration Testing

The purpose of integration tests within this project is to verify and validate the communications between
the different components of the project (i.e. web app, NodeJS server, Java server).

Integration tests should specify the following:

- The input or action of the test.
- The expected result.

Acceptance Testing

Acceptance testing should detail every scenario a user could find themselves, as per the use cases defined in the functional specification and use cases thought of during the course of the development of the project.
An ideal way to document these test cases would be to take inspiration from the 'Gherkin' syntax (https://github.com/cucumber/cucumber/wiki/Gherkin) - Gherkin is used to help describe software's behaviour in a readable way. An example test case format could have (Gherkin syntax in brackets):

- Preconditions (Given)
- Action required (When)
- Expected Results (Then)

As part of this activity, it should also be noted as to what browsers it has been tested in, along with their version number.

User Evaluation

User evaluation should be completed in a way that can quantify the usability of the system.

A proposed way to carry this out would be to follow the ISO 9126-4 Approach.
This approach focuses on three main 'usability metrics' that quantify the usability of the system:

- Effectiveness
- Efficiency
- Satisfaction

Effectiveness
Firstly, a set of tasks that a user must complete must be defined. The tasks must be defined to correlate to the use cases of the systems - how a user would use the system.

Effectiveness can be measured by observing how many tasks a user completes successfully. A way to show the effectiveness could be represent as a percentage of tasks complete and could be calculated like this:
(Number of successfully completed tasks / Number of total tasks) * 100.
These results collected from multiple users could then be averaged to show a value for effectiveness of the system.

Efficiency
The same set of tasks could be used as used to measure effectiveness, and can be completed at the same time evaluating the effectiveness.

Efficiency can be measured in terms of the time it takes to complete a task. It is important to take into account unsuccessfully completed tasks - specifically the time it took before a user gave up on the task. Efficiency per user for each task could be measured as follows:

Result of Task/Time Taken(in seconds).
where Result of Task equals 1 if task successfully complete, else 0.

These numbers can then be averaged across the corresponding tasks for all users. This will express a result of efficiency in terms of 'goals per second'. The higher the number, the better the efficiency.

Satisfaction
After users attempt to complete each task in the task list, they should be given a questionnaire assessing the ease/difficulty of the task. A 'single ease question' which represents a 10 point scale(e.g. 1=very easy, 10=very difficult) which will allow users to define how challenging they found the task.

There should also be a questionnaire given to each test user at the end of test session. The questionnaire can be used to measure the ease of use of the system. A suggestion would be use the 'System Usability Scale' (https://measuringu.com/sus/) to measure the overall satisfaction of the user.

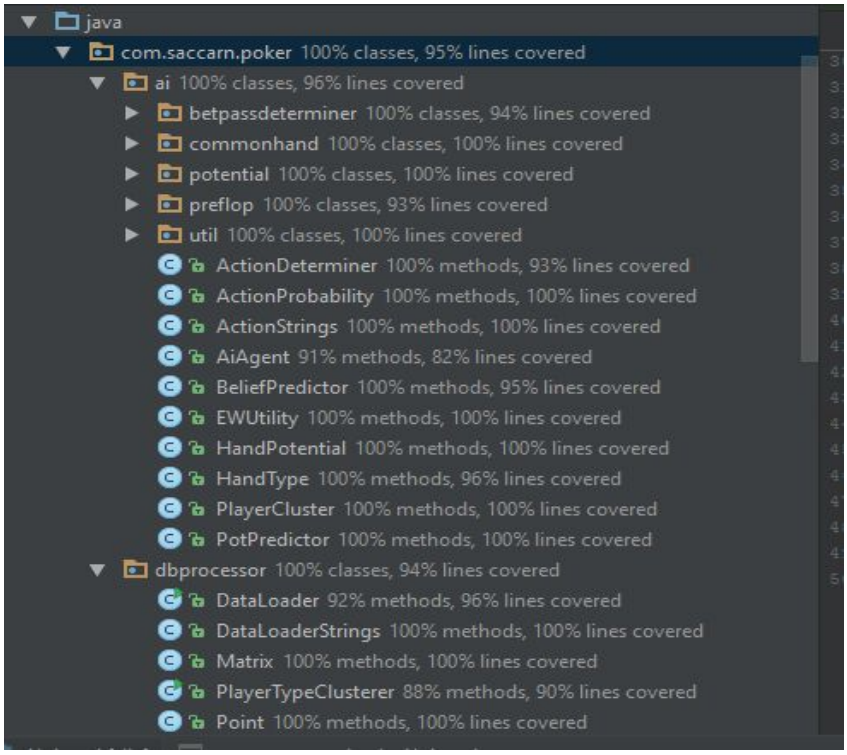
These questionnaires will help pinpoint areas in need of improvement in terms of UI/UX. Users should also be able to describe any comments they have on the system, as they may bring up ideas on how to improve the overall UI/UX.

Unit Testing

AI_Agent
The main focus of unit tests were on the main AI_Agent module. The primary reason for this is to monitor and limit the impact had by any changes made within this module (e.g. supplying different parameters etc.). The tests were run using the JUnit framework.
All unit tests have an assert statement as part its test. The assert statement describes the reason for failure of a test. It also serves as documentation to the test.

Code coverage was generated using Intellij surefire reporting tool. This tool also generates HTML reports documenting which individual lines were covered as part of the unit tests.

Results of Unit Tests
Tests were run using the Intellij JUnit test runner and Maven:
Num Tests: 147. **Num Tests Passed**: 147. **Percentage of num tests passed**: 100%.
Code Coverage Results



Unit Tests for UI
Currently, there are no unit tests for the UI. This has been attempted to been compensated by a high number of acceptance tests.

Unit Tests for PokerHandsProcessor
Currently there is a very low number of unit tests for this module. This is because of the high dependency of parsing files. This module should be refactored in order to use more generic objects (such as InputStream instead of File), in order to make it easier to mock test data for unit testing.
There are currently 16 unit tests for this module, all of which are passing. It has an overall line coverage of 23%.

Integration Testing

The integration test cases are manual tests. As part future contributions to this project, these could be automated. The test cases describe the communications from one module to another

Communication from UI (Web App) to Node Server

Test Case	Actions	Expected Result	Pass (Y/N)
Access UI with Node Server running	Access url: localhost:3000 with web browser	Brings you to 'home' page of UI.	Y
Communication from UI to Node server- attempt to log in	Go to localhost:3000 Attempt to log in using username 'neil'	Node Server should receive log in request. Data should be a JSON object include: socket, username logged in with.	Y
Communication from UI to Node server- attempt to fold	Go to localhost:3000 Attempt to log in using username 'neil' Attempt to fold the hand	Node Server should receive action message. Data received should be JSON object including: socket, username, round of play and action= 'fold'. The opponent model record for the user name should be updated in MongoDB.	Y
Communication from UI to Node server- attempt to check	Go to localhost:3000 Attempt to log in using username 'neil' Attempt to check the hand	Node Server should receive action message. Data should include: socket, username, cards, board cards, round of play and action= 'check'. The opponent model for user name should be retrieved from MongoDB through Mongoose	Y
Communication from UI to Node server- attempt to call	Go to localhost:3000 Attempt to log in using username 'neil' Attempt to call the hand	Node Server should receive action message. Data should include: socket, username, cards, board cards, round of play and action= 'call'. The opponent model for user name should be retrieved from MongoDB through Mongoose	Y
Communication from UI to Node server- attempt to bet	Go to localhost:3000 Attempt to log in using username 'neil' Attempt to bet the hand with 300 chips	Node Server should receive action message. Data should include: socket, username, cards, board cards, round of play, action= 'bet' and amountBet:300. The opponent model for user name	Y

		should be retrieved from MongoDB through Mongoose	
Communication from UI to Node server- attempt to raise	Go to localhost:3000 Attempt to log in using username 'neil' Attempt to raise the hand with 300 chips	Node Server should receive action message. Data should include JSON object with: socket, username, cards, board cards, round of play, action= 'raise' and amountBet:300. The opponent model for user name should be retrieved from MongoDB through Mongoose.	Y
Communication from UI to Node Server - showdown	Go to localhost:3000 Attempt to log in using username 'neil' Go to showdown.	Node Server should receive JSON object to evaluate the hands: Hole Cards, User Cards, Board Cards, Username, socket	Y

Communication from Node Server to UI

Test Case	**Actions**	**Expected Result**	**Pass (Y/N)**
Accepting log in	Send login accepted socketIO message to specific socket	UI moved on from login page to main gameplay screen	Y
Rejecting log in	Send login rejected socketIO message to specific socket	Message on UI appears informing that the log in has been rejected	Y
Sending AI action - fold	Send AI action 'fold' socketIO message to specific socket	The hand is folded by the AI.	Y
Sending AI action - check	Send AI action 'check' socketIO message to specific socket	The hand is checked by the AI.	Y
Sending AI action - fold	Send AI action 'call' socketIO message to specific socket	The hand is called by the AI.	Y
Sending AI action - bet	Send AI action 'bet' (bet1, bet2, bet3) socketIO message to	The hand is bet by the AI. The amount is determined as follows: - Bet1 = 1* ¾ current pot	Y

	specific socket	- Bet2 = 2 * ¾ current pot - Bet2 = 3 * ¾ current pot OR if the AI stack size is smaller, the value of the AI stack	
Sending AI action - raise	Send AI action 'raise' (raise1, raise2, raise3) socketIO message to specific socket	The hand is bet by the AI. The amount is determined as follows: - Raise1 = 1* ¾ current pot - Raise2 = 2 * ¾ current pot - Raise3 = 3 * ¾ current pot PLUS the amount to call the bet OR if the AI stack size is smaller, the value of the AI stack	Y
Sending winner at showdown	Send socketIO message informing user if they have won at showdown to specific socket	Depending on the winner at showdown, the winner of the hand should be displayed at showdown. The winning hand type should also be displayed.	Y

Java Server to Node Server

Test Case	**Actions**	**Expected Result**	**Pass (Y/N)**
Receiving action response from Java Server	Java Server sends message with following info: Username, Action	Node Server receives the message (prints to console).	Y

Node Server to Java Server

Test Case	**Actions**	**Expected Result**	**Pass (Y/N)**
Receiving inputs from Node Server	Node Server sends message with following inputs: Username, cards, board cards, round, opponent model, previous action, amount bet	Java Server receives this (print to console) and creates new thread to deal with determining an action.	Y
Receiving malformed inputs	Node server sends malformed inputs	Java Server receives this (print to console), creates new thread and deals with it gracefully - sends error message	N

		back to node server.	

User Acceptance Tests

There is a high amount of user acceptance tests due to lack of UI unit tests. The test cases describe the functionality of the web app itself regarding how a user would use it.

Feature: Login

Scenario	Preconditions	Action	Result	Pass
User attempts to log in with valid username	User types valid username into username input box.	User clicks enter or presses enter on keyboard	User should be brought to a game screen where a game has started.	**Y**
User attempts to log in with no user name	- User has typed not typed a username into input box.	User clicks enter or presses enter on keyboard	A modal screen should appear informing the user that they have submitted an invalid username and prevent them from logging in.	**Y**
User attempts to log in with a username that is already logged in.	- User types valid user name that is already logged in to the web app.	User clicks enter or presses enter on keyboard	A modal screen should appear informing the user that they have submitted a username that is already logged in and prevent them from logging in.	**Y**
User attempts to log in with a user name with ':' in it	- User types user name containing the colon character	User clicks enter or presses enter on keyboard	A modal screen should appear informing the user that they have submitted a username that is invalid and prevent them from logging in.	

Feature: Initial Game State

Scenario	Preconditions	Action	Result	Pass
A player should have the dealer button	- User should be successfully logged in. - User is on the main game screen	**N/A**	One of either the A.I. player or the user should have dealer button.	**Y**
Players should have equal initial stack sizes	- User should be successfully logged in. - User is on the main game screen	**N/A**	Both the A.I player and the user must have the same number of chips in each of their stacks.	**Y**
User should be able to see their cards	- User should be successfully logged in. - User is on the main game screen	**N/A**	The user should be able to see their cards clearly. The user should also only be able to see the back of the A.I.'s cards.	**Y**
User should not be able to see any board cards	- User should be successfully logged in. - User is on the main game screen	**N/A**	The user should not be able to see any board/community cards.	**Y**
User bet options when user has the dealer button	- User should be successfully logged in. - User is on the main game screen - User has the dealer button - User bet options have been enabled	**N/A**	The user should be given the option to: - Raise - Bet - Fold	
User bet options	- User should be successfully logged in. - User is on the main game screen	User waits for a few seconds	User bet options should be clickable and look enabled.	

	- User bet options should be unclickable and look disabled			
User bet options when A.I. has dealer button	- User should be successfully logged in. - User is on the main game screen - User has the dealer button - User bet options have been enabled	**N/A**	The user should be given the option to: - Raise - Bet - Fold	
Blinds have been correctly placed when the user has the dealer button	- User should be successfully logged in. - User is on the main game screen - User has the dealer button	**N/A**	The user should have placed half of the the big blind as contribution to the pot. The A.I should have placed all of the big blind as their contribution to the pot. The pot should be made up of the total contributions by both players	
Blinds have been correctly placed when the A.I has the dealer button	- User should be successfully logged in. - User is on the main game screen - A.I. has the dealer button	**N/A**	The AI should have placed half of the the big blind as contribution to the pot. The user should have placed all of the big blind as their contribution to the pot. The pot should be made up of the total contributions by both players	

Feature: Preflop

Pre conditions for all Preflop scenarios:

- User should be successfully logged in.
- User is on the main game screen

Scenario	Preconditions	Action	Result	Pass
No community cards at any point of pre flop stage			No community cards should appear at any point at this stage regardless action take by user	
User bet options when user has dealer button	- User has the dealer button - User bet options have been enabled	**N/A**	The user should be given the option to: - Raise - Bet - Fold	
User has dealer button and carries out first action of hand - call	- User has dealer button - User has enough chips to call	User clicks action 'Call'	The pot should be correctly calculated to account for the 'called' contribution towards the pot. The A.I should carry out one of the following actions: - Call - Raise - Fold	
User has dealer button and carries out first action of hand - fold	- User has dealer button	User clicks action 'Fold'	The pot should be added to the A.I. chip stack. A new hand should be dealt out.	
User has dealer button and carries out first action of hand - raise	- User has dealer button - User has enough chips to raise	User inputs amount to raise by and clicks action 'Raise'	The raise contribution should be added to the user contribution and to the total pot. The A.I should carry out one of the following actions: - Call - Raise	

			- Fold	
A.I has dealer button and carries out first action of hand - call	- A.I. has dealer button - A.I. has enough chips to call	A.I. carries out action 'Call'	The amount to call by should be added to the AI contribution and the total pot. The user should be given the option to - Bet - Check - Fold	
A.I has dealer button and carries out first action of hand - raise	- A.I has dealer button - A.I has enough to chips to raise	A.I. carries out action 'Raise'	The amount raised, plus the amount needed to call the bet should be added to A.I. contribution and to the total pot. The user should be given the option to - Call - Raise - Fold	
A.I has dealer button and carries out first action of hand - fold	- User has dealer button	A.I carries out action to 'Fold'	The pot should be added to the user chip stack. A new hand should be dealt out.	
AI checks, followed by User checks	The previous AI action is Check	The user checks	The hand should move on to the flop stage	
User checks, followed by AI checking	The previous user action is Check	The AI checks	The hand should move to the flop stage	
User raises, followed by AI calling	The previous user action is raise	The AI calls	The hand should move to the flop stage	
AI raises, followed by user calling	The previous user action is raise	The AI call	The hand should move to the flop stage	
User bets, followed by AI checking	The previous user action is bet	The AI calls	The hand should move to the flop stage	

AI bets, followed by AI calls	The previous user action is bet	The AI calls	The hand should move to the flop stage	

Feature: Flop

Precondition for all Flop scenarios:

- Preflop completed

Scenario	Preconditions	Action	Result	Pass
Community cards			Should be three three community cards visible to the user	
User has the dealer button, no action completed	User has dealer button		User has option to - Check - Bet - Raise	
A.I has dealer button, no action completed	A.I. has dealer button		User does not have the option to carry out any action until A.I carries out one of following actions: - Check - Bet - Raise	
AI checks, followed by User checks	The previous AI action is Check	The user checks	The hand should move on to the turn stage	
User checks, followed by AI checking	The previous user action is Check	The AI checks	The hand should move to the turn stage	
User raises, followed by AI calling	The previous user action is raise	The AI calls	The hand should move to the turn stage	
AI raises, followed by user calling	The previous user action is raise	The AI call	The hand should move to the turn stage	
User bets, followed by AI checking	The previous user action is bet	The AI calls	The hand should move to the turn stage	

AI bets, followed by AI calls	The previous user action is bet	The AI calls	The hand should move to the turn stage	
A.I carries out action - fold		A.I carries out action to 'Fold'	The pot should be added to the user chip stack. A new hand should be dealt out.	
User carries out action - fold		User carries out action to 'Fold'	The pot should be added to the A.I. chip stack. A new hand should be dealt out.	

Feature: Turn

Precondition for all Turn scenarios:
- Flop completed

Scenario	Preconditions	Action	Result	Pass
Community cards			Should be four community cards visible to the user	
User has the dealer button, no action completed	User has dealer button		User has option to - Check - Bet - Raise	
A.I has dealer button, no action completed	A.I. has dealer button		User does not have the option to carry out any action until A.I carries out one of following actions: - Check - Bet - Raise	
AI checks, followed by User checks	The previous AI action is Check	The user checks	The hand should move on to the river stage.	
User checks,	The previous user	The AI checks	The hand should move	

followed by AI checking	action is Check		to the river stage.	
User raises, followed by AI calling	The previous user action is raise	The AI calls	The hand should move to the river stage.	
AI raises, followed by user calling	The previous user action is raise	The AI call	The hand should move to the river stage.	
User bets, followed by AI checking	The previous user action is bet	The AI calls	The hand should move to the river stage.	
AI bets, followed by AI calls	The previous user action is bet	The AI calls	The hand should move to the river stage.	
A.I carries out action - fold		A.I carries out action to 'Fold'	The pot should be added to the user chip stack. A new hand should be dealt out.	
User carries out action - fold		User carries out action to 'Fold'	The pot should be added to the A.I. chip stack. A new hand should be dealt out.	

Feature: River

Precondition for all River scenarios:
- Turn completed

Scenario	Preconditions	Action	Result	Pass
Community cards			Should be five community cards visible to the user	
User has the dealer button, no action completed	User has dealer button		User has option to - Check - Bet - Raise	

A.I has dealer button, no action completed	A.I. has dealer button		User does not have the option to carry out any action until A.I carries out one of following actions: - Check - Bet - Raise	
AI checks, followed by User checks	The previous AI action is Check	The user checks	The hand should move on to the showdown stage.	
User checks, followed by AI checking	The previous user action is Check	The AI checks	The hand should move to the showdown stage.	
User raises, follwed by AI calling	The previous user action is raise	The AI calls	The hand should move to the showdown stage.	
AI raises, followed by user calling	The previous user action is raise	The AI call	The hand should move to the showdown stage.	
User bets, followed by AI checking	The previous user action is bet	The AI calls	The hand should move to the showdown stage.	
AI bets, followed by AI calls	The previous user action is bet	The AI calls	The hand should move to the showdown stage.	
A.I carries out action - fold		A.I carries out action to ‘Fold’	The pot should be added to the user chip stack. A new hand should be dealt out.	
User carries out action - fold		User carries out action to ‘Fold’	The pot should be added to the A.I. chip stack. A new hand should be dealt out.	

Feature: Showdown

Precondition for all Showdown scenarios:
- River stage completed or all in called.

Scenario	Preconditions	Action	Result	Pass
User has better hand rank than AI at showdown (per rules of poker hand ranking)			The pot should be added to the User chip stack. Contributions and the pot should be reset to 0. A new hand should be dealt out.	
A.I. has better hand rank than user (as per rules of poker hand ranking)			The pot should be added to the AI chip stack. Contributions and the pot should be reset to 0. A new hand should be dealt out	

Feature: New hand

Scenario	Preconditions	Action	Result	Pass
User should not be able to see any board cards		New hand dealt	The user should not be able to see any board/community cards.	**Y**
The A.I should have the dealer button	In the previous hand, the user had the dealer button	New hand dealt	The A.I should have the dealer button for this hand	
The user should have the dealer button	In the previous hand, the A.I. had the dealer button	New hand dealt	The user should have the dealer button for this hand	

Blinds have been correctly placed when the user has the dealer button	- User has the dealer button	New hand dealt	The user should have placed half of the the big blind as contribution to the pot. The A.I should have placed all of the big blind as their contribution to the pot. The pot should be made up of the total contributions by both players	
Blinds have been correctly placed when the A.I has the dealer button	- User should be successfully logged in. - User is on the main game screen - A.I. has the dealer button	New hand dealt	The AI should have placed half of the the big blind as contribution to the pot. The user should have placed all of the big blind as their contribution to the pot. The pot should be made up of the total contributions by both players	

Feature: Game Completion

Scenario	Preconditions	Action	Result	Pass
User does not have enough chips to place a big blind in a hand	User does not have chips to place a big blind in a new hand	New hand	Message appears informing user that they have lost, button inviting player to play a new game is shown	
A.I does not have enough chips to place a big blind in	A.I. does not have enough chips to place a big blind in a new	New hand	Message appears congratulating player on their win, button	

a new hand	hand		inviting player to play a new game is shown	

Game Actions

Feature: Bet

Scenario	Preconditions	Action	Result	Pass
User chooses to bet	User has option to bet	- User types amount greater than/equal the minimum bet and less than/equal their own chip stack - User clicks option to bet	Amount is added to both user contribution and the total pot A.I. turn to carry out an action (call, fold or raise).	
A.I chooses to bet	A.I has option to bet A.I has enough chips to bet.	A.I carries out action to bet.	Amount that the AI has bet by is added to AI contribution and also to the total pot.	
User chooses to bet - not enough chips	User has option to bet User has less chips than minimum bet	- User types amount greater than/equal the minimum bet and less than/equal their own chip stack - User clicks option to bet	A modal appears to the user informing them that they must bet a valid amount - between the minimum bet and their chip stack size.	
User chooses to bet - amount over their stack size	User has option to bet	- User types amount greater to their own chip stack - User clicks option to bet	A modal appears to the user informing them that they must bet a valid amount - between the minimum bet and their chip stack size.	
User chooses to bet - amount below minimum	User has option to bet	- User types amount below the minimum bet - User clicks option	A modal appears to the user informing them that they must bet a valid amount -	

		to bet	between the minimum bet and their chip stack size.	
User chooses to bet - invalid number	User has option to bet	- User types an amount that can not be parsed as a positive integer e.g. “test”, “-400”, “forty”. - User clicks option to bet	A modal screen appears to the user informing them that: ‘The amount you have entered is not a valid integer. Please enter valid value.’	

Feature: Fold

Scenario	Preconditions	Action	Result	Pass
User chooses to fold	User has the option to fold	User clicks on ‘fold option’	The pot should be added to the A.I. chip stack. Contributions and the pot should be reset to 0. A new hand should be dealt out.	
A.I. chooses to fold	A.I. has the option to fold	User clicks on ‘fold’ option	The pot should be added to the user chip stack. Contributions and the pot should be reset to 0. A new hand should be dealt out.	

Feature: Raise

Scenario	Preconditions	Action	Result	Pass
User chooses to raise	User has option to raise User has enough	- User types amount into raise by input greater than/equal	Both the amount to call by and amount to raise by is added to	

	chips to both call and raise a bet.	the minimum bet and less than/equal their own chip stack - User clicks option to raise	both user contribution and the total pot A.I. turn to carry out an action (call, fold or raise).	
A.I chooses to raise	A.I has option to raise A.I has enough chips to both call and raise a bet.	A.I carries out action to raise.	Amount that the AI has raise by is added to AI contribution and also to the total pot. User turn to carry out action (call, fold, or raise)	
User chooses to raise- not enough chips	User has option to raise User has raise chips than minimum bet	- User types amount greater than/equal the minimum bet and less than/equal their own chip stack - User clicks option to raise	A modal appears to the user informing them that they must bet a valid amount - between the minimum bet and their chip stack size.	
User chooses to raise - amount over their stack size	User has option to raise	- User types amount greater to their own chip stack - User clicks option to raise	A modal appears to the user informing them that they must bet a valid amount - between the minimum bet and their chip stack size.	
User chooses to raise- amount below minimum	User has option to raise	- User types amount below the minimum bet - User clicks option to raise	A modal appears to the user informing them that they must bet a valid amount - between the minimum bet and their chip stack size.	
User chooses to raise - invalid number	User has option to raise	- User types an amount that can not be parsed as a positive integer e.g) “testraise”, “-400”, “forty” - User clicks option to raise	A modal screen appears to the user informing them that: ‘The amount you have entered is not a valid integer. Please enter valid value.’	

Feature: Check

Scenario	Preconditions	Action	Result	Pass
User chooses to check	User has option to check	- User clicks option to check	No contribution is added to either user contribution or the total pot A.I. turn to carry out an action.	
A.I. chooses to check	A.I. has option to check	A.I. carries out option to check	No contribution is added to either A.I. contribution or the total pot User turn to carry out action	

Feature: Call

Scenario	Preconditions	Action	Result	Pass
User chooses to call	User has option to call User has enough to chips to call	- User clicks option to call	The amount called is the contribution is added to both user contribution and the total pot A.I. turn to carry out an action.	
A.I. chooses to call	A.I. has option to check A.I has enough chips to call	A.I. carries out option to call	The amount called is the contribution is added to both user contribution and the total pot User turn to carry out action	
User chooses to call - not enough chips	User has option to call User does not have	- User clicks option to call	The game ends with winner being: - A.I	

	enough to chips to call			
A.I. chooses to call - not enough chips	A.I. has option to call A.I has not enough chips to call	A.I. carries out option to call	The game ends with winner being: - User	

Feature: All In

All In - extends bet/raise features

Scenario	Preconditions	Action	Result	Pass
User clicks on all in - AI and User have equal amount of chips	AI and user have equal amount of chips	User clicks option to go All In.	The amount that the user bet all in should be removed from the user stack (i.e. stack should be zero). The contribution should be added to the total pot.	
User clicks on all in - User has greater amount of chips than AI	User has more chips than the AI	User clicks option to go All In.	The amount that the AI has should be removed from the user stack. The contribution should be added to the pot.	
User clicks on all in - AI has greater amount of chips than User	AI has more chips than the User	User clicks option to go All In.	The amount that the User has should be removed from the user stack (i.e. user stack should be zero). The contribution should be added to the pot.	

Game Objects

Feature: Cards

Scenario	Preconditions	Action	Result	Pass
Valid Card - value			Valid card should one value of the following: - 2,3,4,5,6,7,8,9, 10,J,Q,K,A	
Valid Card - suit			Valid card should have one suit of one of the following: - Hearts, Spades, Clubs, Diamonds	
Valid Cards in a hand			No two cards equivalent in both suit and value may appear in one hand.	

Feature: Stacks

Scenario	Preconditions	Action	Result	Pass
Valid Stack			Stack should never have a negative value. (always >= 0)	

Feature: Minimum Bet

Scenario	Preconditions	Action	Result	Pass
Minimum should increase every ‘X’ hands		‘X’ hands played	The minimum bet should double	
The number of		A hand is played	The number of hands	

hands should decrease for every hand that passes, apart from '0 hands left' case			left until the min. bet doubles should decrement by one.	
When there are 0 hands left, the number of hands should then go back to a higher number(e.g. 10/15/20)	1 hand left until minimum bet doubles	A hand is played	The number should reset to a higher number.	

Web Browser Testing

The web app has been tested and validated in the following web browsers/versions.

Web Browser	**Version No.**
Google Chrome	58
Mozilla Firefox	49
Microsoft Edge	38

User Evaluation

The tasks were designed to attempt to test all use cases/ functionality of the system. They were written in a manner to set up a context for the test user. The tasks for user evaluation were defined as follows:

User Task Evaluation - Tasks

Task # 1
You want to play some poker online. Log in to the system.

Task # 2
The AI has checked into you. You wish to continue playing into the next stage of play but do not wish to add any chips to the pot.

Task # 3
You have a hand of medium strength. You wish to increase the stakes by adding 400 chips to the pot.

Task # 4
The AI has bet into you. You do not wish to continue playing this hand. End this current hand.

Task # 5
You have just checked into the AI. You want to identify what action they perform in response.

Task # 6
You have a strong hand. You wish to bet the entirety of your chip stack into the pot.

Task # 7
You and the AI have gone to showdown. You wish to identify who has won the hands and what cards the AI had.

Effectiveness

Efficiency

Satisfaction
The following questionnaire was given out to measure satisfaction. It uses slightly adapted questions from the 'System Usability Scale' suggested in the test plan.

Questions were accompanied with scale marked from 1 - 5.

1. If I wanted to play poker online, I think that I would like to use this system frequently.
2. I found the system unnecessarily complex.
3. I thought the system was easy to use.
4. I think that I would need the support of a technical person to be able to use this system.
5. I found the various functions in this system were well integrated.
6. I thought there was too much inconsistency in this system.
7. I would imagine that most people would learn to use this system very quickly.
8. I found the system very cumbersome to use.
9. I felt very confident using the system.
10. I needed to learn a lot of things before I could get going with this system.

System usability test scores:

Appendix B

Validation - Betting Parameters

These test cases can be found and run in the validation module of the project.
The results are based two AI agents playing against each other, with 20,000 hands played, per test case.

Process used to determine optimal values for parameters:

- Define a default range of values for the betting parameters.
- For each parameter, test against different value for that particular parameters and obtain value which seems to be performing the best.
- Create a new set of values for the parameters from these proposed values and test against previous default parameters.

The following series of tests are the tests ran and documented to produce the set of parameters that are used within the project.
The test classes can be found in the com.saccarn.poker.testbetpassvalues package in the validation module of the project. The values can be found in com.saccarn.poker.betpassvalues package of the same module.

The recorded results include the winning player, defined in the test case, the winning margin in terms of big blinds, and the variance of this margin. This also applies to Appendices C, D, E.

Test against random distribution of parameters to determine default set of parameters

Test Case	Test Class (to run)	Values	Result
Default Values (Player One) vs BetPassValuesTest1 (Player Two)	TestDefaultVsBetPassValues1	DefaultValues BetPassValuesTest1	Player One won: 1.372 big blinds per hand +- 0.2354
Default Values (Player One) vs BetPassValuesTest2 (Player 2)	TestDefaultVsBetPassValues2	DefaultValues BetPassValuesTest2	Player One won: 1.4839 big blinds per hand +- 0.37847

Conclusion: Default Values seem good enough to try and improve on.

Pass parameter

Test Case	Test Class (to run)	Values	Result

Default Values (Player One) vs BetPassValuesTest3 (Player 2)	TestDefaultVsBetPassValues3	DefaultValues BetPassValuesTest3	Player One won: 0.14175 big blinds per hand +- 0.21417
Default Values (Player One) vs BetPassValuesTest4 (Player 2)	TestDefaultVsBetPassValues4	DefaultValues BetPassValuesTest4	Player One won: 0.07745 big blinds per hand +- 9.14908

Conclusion: Default Pass Parameter seems to produce optimal results compared to other values

Bet2 parameter

Test Case	Test Class (to run)	Values	Result
Default Values (Player One) vs BetPassValuesTest5 (Player 2)	TestDefaultVsBetPassValues5	DefaultValues BetPassValuesTest5	Player Two won: 0.018 big blinds per hand +- 0.1268 Player One won : 0.3248
Default Values (Player One) vs BetPassValuesTest6 (Player 2)	TestDefaultVsBetPassValues6	DefaultValues BetPassValuesTest6	Player Two won: 0.1768 big blinds per hand Player One won: 0.482 +- 0.08276
BetPassValuesTest5 (Player One) vs BetPassValuesTest6 (Player Two)	TestBetPassValues5VsBetPassValues6	BetPassValues5 BetPassValues6	Player One won 0.2526 big blinds per hand +- 0.191511.
BetPassValuesTest5 (Player One) vs BetPassValuesTest7 (Player Two)	TestBetPassValues5VsBetPassValues7	BetPassValues5 BetPassValues7	Player One won: 0.344435 big blinds per hand +- 0.29577
BetPassValuesTest6 (Player One) vs BetPassValuesTest7 (Player Two)	TestBetPassValues6VsBetPassValues7	BetPassValues6 BetPassValues7	Player Two won: 0.001 big blinds per hand +- 0.2146
BetPassValuesTest5 (Player One) vs BetPassValuesTest8 (Player Two)	TestBetPassValues5VsBetPassValues8	BetPassValuesTest5 BetPassValuesTest8	Player Two won: 0.0095 big blinds per hand. 0.3748 +- 0.2327 (Player one)

BetPassValuesTest8 (Player One) vs BetPassValuesTest9 (Player Two)	TestBetPassValues8VsBetPassValues9	BetPassValuesTest8 BetPassValuesTest9	Player One won: 0.19065 big blinds per hand +- 0.2878
BetPassValuesTest8 (Player One) vs BetPassValuesTest10 (Player Two)	TestBetPassValues8VsBetPassValues10	BetPassValues8 BetPassValues10	Player One won: 0.2283 big blinds per hand. 0.058 +- 0.1753

Conclusion: Bet2 Parameter of BetPassValuesTest8 seems to produce optimal results

Bet3 Parameter

Test Case	Test Class (to run)	Values	Result
DefaultValues (Player One) vs BetPassValuesTest11 (Player Two)	TestDefaultVsBetPassValues11	DefaultValues BetPassValues11	Player One won: 0.3951 big blinds per hand +- 0.1649.
DefaultValues (Player One) vs BetPassValuesTest12 (Player Two)	TestDefaultVsBetPassValues12	DefaultValues BetPassValues12	Player One won: 0.0575 big blinds per hand +- 0.20893.
DefaultValues (Player One) vs BetPassValuesTest13 (Player Two)	TestDefaultVsBetPassValues13	DefaultValues BetPassValues13	Player One won: 0.0946 big blinds per hand +- 0.1465.

Conclusion: Default Pass Parameter seems to produce optimal results compared to other values.

All In Parameter

Test Case	Test Class (to run)	Values	Result
DefaultValues (Player One) vs BetPassValuesTest14	TestDefaultVsBetPassValues14	DefaultValues BetPassValuesTest14	Player One won: 0.015173 big blinds per hand +- 0.14198.
DefaultValues (Player One) vs BetPassValuesTest15 (Player Two)	TestDefaultVsBetPassValues15	DefaultValues BetPassValuesTest15	Player One won: 0.1109 big blinds per hand +- 0.157876.

DefaultValues (Player One) vs BetPassValuesTest16 (PlayerTwo)	TestDefaultVsBetPassValues16	DefaultValues BetPassValuesTest16	Player Two won: 0.07265 big blinds per hand +- 0.08768
BetPassValuesTest15 (Player One) vs BetPassValuesTest16 (Player Two)	TesDefaultVstBetPassValues15VsBetPassValues16	BetPassValuesTest15 BetPassValuesTest16	Player Two won: 0.223845 big blinds per hand +- 0.12841.
BetPassValuesTest14 (Player One) vs BetPassValuesTest16 (Player Two)	TestBetPassValues14VsBetPassValues16	BetPassValuesTest14 BetPassValuesTest16	Player Two won: 0.202245 big blinds per hand. +- 0.0736

Conclusion: All in Parameter in BetPassValuesTest16 seems to produce optimal results compared to other values.

Obtained values integrated together:

Test Case	**Test Class (to run)**	**Values**	**Result**
DefaultValues (Player One) Vs ProposedDefaultValues 1 (Player Two)	TestDefaultValuesVsProposedDefaultValues	DefaultValues ProposedDefaultValues	Player Two won: 0.1066 big blinds per hand +- 0.1895.

Conclusion: New Proposed values beats the previous default values and should be used in the project.

Appendix C

Validation - Testing Common Hand influencing belief vs. Common Hand Not influencing belief

These test cases can be found and run in the validation module of the project.
The results are based two AI agents playing against each other, with 20,000 hands played, per test case.

The test classes can be found in the com.saccarn.poker.testshandpotential and com.saccarn.poker.testscommonhandvalues package in the validation module of the project. The values can be found in com.saccarn.poker.commonhandvalues and com.saccarn.poker.handpotentialstraightvalues package of the same module.
The recorded results include the winning player, defined in the test case, the winning margin in terms of big blinds, and the variance of this margin.

Test Case	Test Class (to run)	Values	Result
Using CommonHand class influence (Player 1) vs NO CommonHand class influence (Player 2)	TestCommonHandTrue VsCommonHandFalse	CommonHandValues.COMMON_HAND_TRUE, CommonHandValues.COMMON_HAND_FALSE	Player Two won: 0.2657 big blinds per hand +- 0.104236

Validation - Testing Hand Potential influencing belief vs. Hand Potential Not influencing belief

These test cases can be found and run in the validation module of the project.
The results are based two AI agents playing against each other, with 20,000 hands played, per test case.

Partial Straight/Flush recognition vs no partial Straight/Flush recognition

Test Case	Test Class (to run)	Values	Result
Using HandPotential class influence (Player 1) vs NO HandPotential class influence (Player	TestHandPotentialTrue VsHandPotentialFalse	HandPotentialValues.HAND_POTENTIAL_TRUE, HandPotentialValues.H	Player One won: 0.28625 big blinds per hand +- 0.161786

2)		AND_POTENTIAL_FALSE	

Appendix D

Opponent Model Evaluation

These test cases can be found and run in the validation module of the project.
The results are based two AI agents playing against each other, with 20,000 hands played, per test case.

The test classes can be found in the com.saccarn.poker.testsopponentmodels package in the validation module of the project. The values can be found in com.saccarn.poker.opponentmodels package of the same module.

The recorded results include the winning player, defined in the test case, the winning margin in terms of big blinds, and the variance of this margin.

Test Case	**Test Class (to run)**	**Values**	**Result**
Default Opponent Model (Player 1) vs 'Tight' Opponent Model (Player 2)	TestDefaultVsTightOpponentModel	Default Opponent Model Tight Opponent Model	Player 2 won: 0.114848 big blind per hand +- 0.14597.
Default Opponent Model (Player 1) vs 'Loose' Opponent Model (Player 2)	TestDefaultVsLooseOpponentModel	Default Opponent Model Loose Opponent Model	Player 1 won: 0.85546 big blinds per hand +- 0.2127229.
Default Opponent Model (Player 1) vs 'Unusual' Opponent Model (Player 2)	TestDefaultVsUnusualOpponentModel	Default Opponent Model Unusual Opponent Model	Player 1 won: 0.1326 big blinds per hand +- 0.1019436
Default Opponent Mode (Player 1) vs 'Cluster 1' Opponent Model	TestDefaultVsCluster1Model	Default Opponent Model Cluster 1 Opponent Model	Player 1 won: 0.3749 +- 0.16 big blinds per hand

Default Opponent Model (Player 1) vs 'Cluster 2' Opponent Model (Player 2)	TestDefaultVsCluster2 Model	Default Opponent Model Cluster 2 Opponent Model	Player 1 won: 0.087 big blinds per hand +- 0.1548
Default Opponent Model (Player 1) vs 'Cluster 3' Opponent Model (Player 2)	TestDefaultVsCluster3 Model	Default Opponent Model Cluster 3 Opponent Model	Player 1 won: 0.6269 +- 0.1059 big blinds per hand

Appendix E

Validation - Preflop values

Similar strategy used to determine values as for BetPassValues.

The test classes can be found in the com.saccarn.poker.testspreflopvalues package in the validation module of the project. The values can be found in com.saccarn.poker.preflopvalues package of the same module.

PreFlop Rank Value Parameter

Test Case	Test Class (to run)	Values	Result
Default Preflop Values (Player 1) vs PreFlopValuesTest1 (Player 2)	TestDefaultVsPreFlopValuesTest1	DefaultValues PreFlopValuesTest1	Player 1 won: 0.2908 big blind per hand +- 0.1549.
Default Preflop Values (Player 1) vs PreFlopValuesTest2 (Player 2)	TestDefaultVsPreFlopValuesTest2	DefaultValues PreFlopValuesTest2	Player 1 won: 0.29225 big blinds per hand +- 0.2586.
Default Preflop Values (Player 1) vs PreFlopValuesTest3 (Player 2)	TestDefaultVsPreFlopValuesTest3	DefaultValues PreFlopValuesTest3	Player 1 won: 0.06945 big blinds per hand +- 0.0898.
Default Preflop Values (Player 1) vs PreFlopValuesTest4	TestDefaultVsPreFlopValuesTest4	DefaultValues PreFlopValuesTest4	Player 1 won: 0.1838 big blinds per hand +- 0.07141563.

(Player 2)			
Default Preflop Values (Player 1) vs PreFlopValuesTest5 (Player 2)	TestDefaultVsPreFlopValuesTest5	DefaultValues PreFlopValuesTest5	Player 1 won: 0.1838 big blinds per hand +- 0.7141563.

PreFlop RandomFold Value Parameter

Test Case	Test Class (to run)	Values	Result
Default Preflop Values (Player 1) vs PreFlopValuesTest6 (Player 2)	TestDefaultVsPreFlopValuesTest6	DefaultValues PreFlopValuesTest6	Player 1 won: 0.0851 big blinds per hand +- 0.15556.
Default Preflop Values (Player 1) vs PreFlopValuesTest7 (Player 2)	TestDefaultVsPreFlopValuesTest7	DefaultValues PreFlopValuesTest7	Player 1 won: 0.02899 big blinds per hand +- 0.15367.
Default Preflop Values (Player 1) vs PreFlopValuesTest8 (Player 2)	TestDefaultVsPreFlopValuesTest8	DefaultValues PreFlopValuesTest8	Player 1 won: 0.16905 big blinds per hand +- 0.248.
Default Preflop Values (Player 1) vs PreFlopValuesTest9 (Player 2)	TestDefaultVsPreFlopValuesTest9	DefaultValues PreFlopValuesTest9	Player 1 won: 0.13625 big blinds per hand +- 0.10431.
Default Preflop Values (Player 1) vs PreFlopValuesTest10 (Player 2)	TestDefaultVsPreFlopValuesTest10	DefaultValues PreFlopValuesTest10	Player 1 won: 0.13755 big blinds per hand +- 0.15600.
Default Preflop Values (Player 1) vs PreFlopValuesTest11 (Player 2)	TestDefaultVsPreFlopValuesTest11	DefaultValues PreFlopValuesTest11	Player 1 won: 0.150755 big blinds per hand +- 0.15942.

Conclusions: Original values for pre flop values seem to produce optimal results.