

National College of Ireland
BSc in Computing
2017/2018

Lee Murray
X14538387
X14538387@student.ncirl.ie

Heads-Up Hold'em

Technical Report



Declaration Cover Sheet for Project Submission

Name: Lee Murray
Student Number: x14538387
Supervisor: Vikas Sahni

Section 2 Confirmation of Authorship

The acceptance of your work is subject to your signature on the following declaration: I confirm that I have read the College statement on plagiarism (summarized overleaf and printed in full in the Student Handbook) and that the work I have submitted for assessment is entirely my own work.

Signature: Lee Murray

Date: 13/05/2018

Table of Contents

Executive Summary	6
1 Introduction	7
1.1 Background.....	7
1.2 Aims	11
1.3 Technologies	12
1.4 Structure	13
1.5 System.....	13
1.6 Requirements	13
1.6.1 Functional requirements.....	13
1.6.2 Use Case Diagram	14
1.6.3 Requirement 1 <User Registration>	14
1.6.4 Requirement 2 <User Login>	17
1.6.5 Requirement 3 <Betting Poker Configuration>	19
1.6.6 Requirement 3 <Appearance Poker Configuration>.....	21
1.6.7 Requirement 3 <Play Poker Menu>	23
1.6.8 Requirement 3 <Play Versus AI>	25
1.7 User Requirements	27
1.7.1 User requirements.....	27
1.7.2 Environmental requirements	27
1.7.3 Usability requirements.....	28
1.8 Non-Functional Requirements	28
1.8.1 Performance/Response time requirement.....	28
1.8.2 Availability requirement	29
1.8.3 Data requirements.....	29
1.8.4 Security requirement	29
1.8.5 Reliability requirement.....	29
1.8.6 Maintainability requirement	30
1.8.7 Portability requirement	30
1.8.8 Extendibility requirement.....	30
1.9 Design and Architecture.....	32
1.10 Implementation	33

1.11	Graphical User Interface (GUI) Layout.....	44
1.12	Testing.....	52
1.13	Evaluation	56
1.14	Definitions, Acronyms, and Abbreviations.....	57
2	Conclusions	60
3	Further development or research.....	62
4	Bibliography	63
5	Appendix.....	68
5.1	Project Proposal	68
5.2	Background.....	69
5.3	Technical Details	73
5.4	Project Plan	75
5.5	Monthly Journals.....	75

Executive Summary

In today's world, Artificial Intelligence exists in every game we play. It was a challenge for computer scientists to beat the professional players since computers were invented. Many games are deterministic perfect information games like chess or checkers where there is no chance and there is no any hidden information from the opponent. This thesis investigates non-deterministic imperfect information games like poker that are very popular in real world. There exists a technique that calculates the strategies over time to win the player by reaching the Nash equilibrium.

Poker is currently the world's most played card game. Hundreds of thousands of people play poker every day, and can play in a real-life environment or over the internet using a distributed application running a simulation of the game. One of the biggest reasons for poker's recent success is its fundamental dynamics. The 'hidden' elements of the game means players must observe their opponent's characteristics to be able to arrive at good decisions, given their options. A very good poker player will consistently dominate a sub-optimal opponent, although stochastic elements apply heavy statistical variation to the game, allowing weak players to win occasionally.

The game of poker offers a well-defined domain in which to investigate some fundamental issues in computing science, such as how to handle deliberate misinformation, and how to make intelligent guesses based on partial knowledge. This project will aim to investigate what Artificial Intelligence techniques can be applied to the domain in order to play up to a human standard of decision making.

The findings of the research have application beyond the realm of poker, and can be applied to financial, weather and military domains, or more generally, any domain with a nondeterministic outcome that incorporates stochastic elements.

1 Introduction

1.1 *Background*

In the domain of Artificial Intelligence there has been a plethora of games that have been solved to date. Some examples of these agents are, IBM's "Deep Blue" for chess, The University of Alberta's "Chinook" for checkers and Michael Buro's "Logistello" for Othello.

These agents have effectively solved those games and have beaten the best human minds in the world, demonstrating the power of computational processing. However, what all these games have in common is, they are all "perfect information" games. Perfect information games refer to the game in which each player, at any point in the game has complete knowledge of the current game state. Games like Chess, Checkers and Backgammon are "perfect information" games. Players who play these have perfect knowledge of the game state as they can see all remaining pieces on the game board.

Well known game-search trees, such as alpha-beta search can be used to explore deep into the game tree to find and choose the worse-case action the opponent cannot compete against.

In contrast to this, Poker is a "imperfect information" game, this means that certain information within the game is private, in terms of poker, each player receives private cards. As a result, no player can know the current position in the game tree.

Poker is a non-deterministic game. A player's actions within the poker domain can never guarantee the same outcome.

Poker has stochastic outcomes. This element of change through random shuffling of the cards creates uncertainty, and adds a great deal of variance to the results.

Texas Hold'em is a poker variation that uses community cards. This variant of Poker was chosen because its rules have specific characteristics that allow new developed methodologies to be adapted to other Poker variations with reduced effort.

Rules

At the beginning of every game, two cards are dealt to each player. The dealer player is assigned and marked with a dealer button. The dealer position rotates clockwise from game to game. After that, the two players to the left of dealer post the blind bets. The first player is called small blind, and the other one is called big blind. They respectively post half of minimum and the minimum bet. The player that starts the game is the one on the left of the big blind. One example of an initial table configuration is shown in Figure 2. The dealer is the player at seat F and the small and big blind players are respectively the A and B seats.

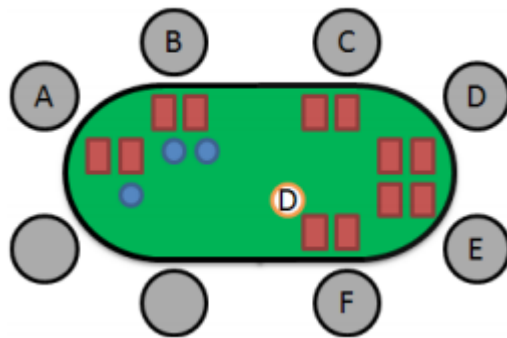


Figure 1: Table Structure

Table Layout

The first player to act is the player to the left of the big blind (Player C) And the next player is the closest one to the left of the current player. Each player can choose one of the following actions

- Call: Match the current highest bet
- Raise: Bet higher than the current highest bet
- Fold: Forfeit the hand, thus give up the pot

There are four betting rounds in Texas Hold'em, where each round new community cards are revealed.

- Pre-Flop: No community cards
- Flop: three community cards revealed

- Turn: The fourth community card is revealed
- River: The fifth community card is revealed

After the river, if at least 2 players agree to call the pot, the showdown round comes. This is when all players may show their cards and the one with the best hand wins the pot. If players have similar ranked hands, there is a tie and the pot is divided. This is otherwise known as a “chop-pot”

Hand Rankings

A poker hand is a set of five cards that identifies the score of a player in poker. The hand is made by combining the player’s personal cards with the community cards. The table below presents the ranking of each combination with a short description.

Hand Description	Hand Example
Royal Flush: this is the best possible hand in standard five-card Poker. Ace, King, Queen, Jack and 10, all of the same suit.	
Straight Flush: Any five-card sequence in the same suit.	
Four of a Kind: Any set with four cards with the same rank.	
Full House: Three cards with the same rank plus two cards with the same rank.	
Flush: Any set with five cards of the same suit, but not in sequence.	
Straight: Five cards in sequence, but with different suit.	
Three of a kind: three cards with the same rank.	
Two Pair: Two separate pairs, and one kicker of different value. The kicker is used to decide upon a tie of the same two pairs.	
One Pair: Two cards with the same rank and three kicker cards.	
High Card: Any hand that does not qualify as one of the better hands above. Ranked by top card, then the second card and so on.	

Figure 2: Hand Examples

Hand Evaluation Algorithms

The algorithm in which that is used to quantify the agent’s Hand Strength, regardless of all cards being dealt. This algorithm is key, as it considers all the possible better hands the agent could have, the same, and all the worse

hands at the point of calculation. The algorithm iterates through all possible starting hands and returns a percentage as a result.

```
Function HandRank(Hand) {  
    Sort(Hand);  
    If IsStraightFlush(Hand) Return 9;  
    If IsIsFourOfAKind(Hand) Return 8;  
    If IsFullHouse(Hand) Return 7;  
    If IsFlush(Hand) Return 6;  
    If IsStraight(Hand) Return 5;  
    If IsThreeOfAKind(Hand) Return 4;  
    If IsTwoPairs(Hand) Return 3;  
    If IsOnePair(Hand) Return 2;  
    Return 1;  
}
```

Figure 3: Hank Ranks

Hand Potential

Hand Potential is an algorithm that calculates the possible evolution of the hand quality throughout the game. In Texas Hold'em, when the game reaches the Flop round, there are still two more community cards to be revealed. This means that the current hand rank may improve, since the hand is composed of the set of five available cards that has the highest rank among all available cards. This is an extension of the hand evaluation, but instead of only considering the current available cards, it considers the possible community cards that have not been revealed yet. This also considers that the opponent's hands might improve as well.

Non-Deterministic Game

Non-deterministic games are often described as games with an element of chance. These games do not result in predictable outcome. Examples of such games are Backgammon and Poker (their source of chance is dice and card respectively). What makes these games different from deterministic games are the additional nodes called 'Chance' or 'Nature' in their game trees.

Imperfect Information

Imperfect information game corresponds to the game in which certain information is private, meaning that other players cannot see it. For example, in Poker each player 10 | P a g e receives private cards. As a result of this, no player can clearly know the current position in the game tree.

The utility or payoff

The utility in the game is the expected value when a round of a game is played. In the poker game, it is the number of chips that was acquired or lost at the end of the hand (round).

Nash Equilibrium

Nash equilibrium is a strategy profile σ where no player can increase their utility by unilaterally changing their strategy (Johanson, 2007): This means that for player 1, there is no other strategy in Σ_1 that would produce more utility against σ_2 than its strategy in σ . The same is true of player 2. (Johanson, 2007)

1.2 Aims

The game of poker sets the stage for a well-defined domain that allows for the investigation of various fundamental issues in computer science and artificial intelligence, such as how to handle misinformation and how to develop an intelligent agent to process Responses based on this partial knowledge.

This project will aim to investigate what Artificial Intelligence techniques can be applied in order to,

- Create an agent that will perform on a human standard of decision making that is capable of playing strong no-limit Texas Holdem
- Investigate the characteristics of strong poker playing and compare these results to the agent solution.

- Measure the performance of the agent against human opposition over many hands, and document the results
- Create a fun and interactive experience for the user
- Design an agent to play the no-limit poker variant

1.3 Technologies

Implementation

This application will be developed using an Agile software development process. The specific agile practice I intend to adopt for this project is Iterative Development. The main idea behind Iterative Development is to break down the whole project into smaller parts or iterations. This was chosen with the goal of completing significant parts of the project at the end of each iteration. I believe this will be proving to be beneficial as it allows for the development of the more basic aspects of the project such as implementing the Counterfactual Regret Minimization algorithm before jumping straight into Poker AI development.

Java

Java is the programming language that will be used to develop the application. This is because Java allows applications to be easily integrated with web application using Java Applet, another reason behind using Java was for future extension of the application to be ported to mobile, using Android application development which uses Java. The object orientated nature of Java, helps in separating the features of the application for easier management and debugging.

MigLayout – Java Layout Manager

MigLayout is a grid based layout that allows for designing complex GUI layouts for Swing applications with ease.

MySQL

MySQL was the primary relational database management system taught to this year's 4th Year NCI computing students during their time at college. Thus, it made sense to use MySQL for this project rather than having to learn an entire new database system.

1.4 Structure

The structure of this document is as follows, Requirement which contain functional and non-functional requirements. Design and Architecture, which include UML diagrams, use cases, system architecture, hardware and software architecture diagrams. Implementation, Testing, GUI layout, Customer testing, Evaluation and Conclusion

1.5 System

1.6 Requirements

The requirements were among the first things to be considered at the outset of the Poker AI project. To produce these requirements, I posted on various Poker forums to ask them various questions as what they would like to see in a Poker Bot application. I also read comments reviews of different types of Poker applications seen on the Google Play Store that incorporated an AI agent and took their opinions on what they valued into account.

1.6.1 Functional requirements

- The AI should function as intended
 - This requirement at its most basic level is the core functionality. The AI agent must be able to respond accordingly to the user's turns
- Hand Evaluation
 - Assessing the probability of a hand improving as more community cards appear
- Better Strategy
 - Determine whether to fold, call/check, or bet/raise in any given situation
- Bluffing
 - Allow the AI to make a profit from a weak hand and to create a false impression about your play
- Unpredictability

- Make it difficult for your opponent to form an accurate model of the AI's strategy
- Opponent Modelling
 - Used to determine a likely probability distribution for the opponents hidden cards.
- Registration
 - A new user must be able to sign up to the system
- Login
 - An already registered user must be able to log into the system
- Customise Appearance
 - Choose the style of both the cards and playing table
- Select AI type
 - Choosing between the three types of AI, chump, conservative and optimal play.
- The Poker-AI application must work on mobile devices
 - From market research, the majority of Poker applications are developed for mobile devices

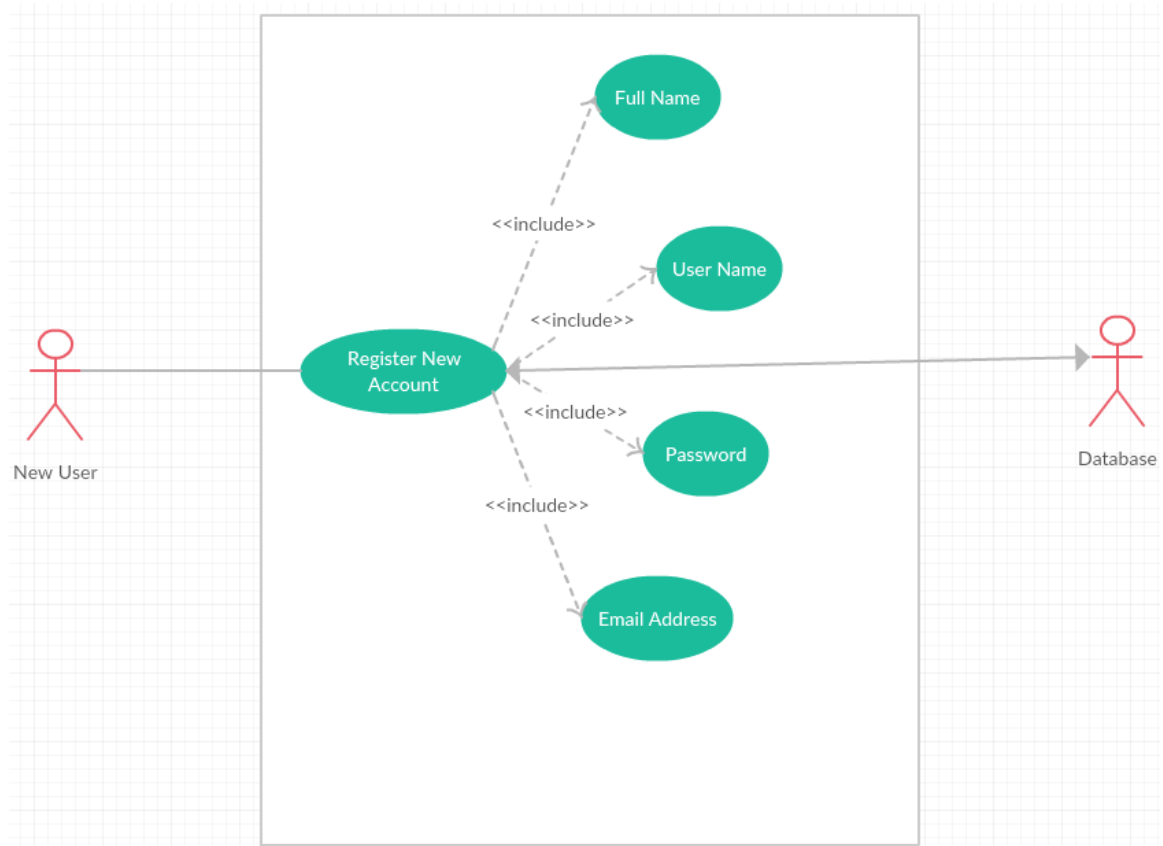
1.6.2 Use Case Diagram

1.6.3 Requirement 1 <User Registration>

1.6.3.1 Description & Priority

This requirement relates to an “unregistered” user who is required to create a new account to become an active “registered” user. This process is crucial as it is required to be a “registered” user to have access to the application.

1.6.3.2 Use Case



Scope

The scope of this use case is to register a new user to the system

Description

This use case describes the registering of a new user to the system, the user is required to make an account to become an active or “Registered User”. This process is crucial as without it, no users can have accounts on the system and thus cannot access the functionality of the application

Flow Description

Precondition

User has not registered an account

Activation

“New User” accesses the application and clicks on the “Registration” button

Main flow

- The user enters all requested information i.e name, email etc
- Application displays a confirmation message that the user has successfully created an account

Alternate flow

Fields not completed

- User has not completed all relevant fields, so the application will highlight all required fields and wait for the user to re-submit with the correct information

Username Already exists

- User supplies a “username” that is already registered. The application will inform the user the username is taken.

Termination

Main flow

- User successfully registers an account

Alternative flow

- User must attempt to register again

Post condition

Main flow

- User is directed to the menu page

Alternative flow

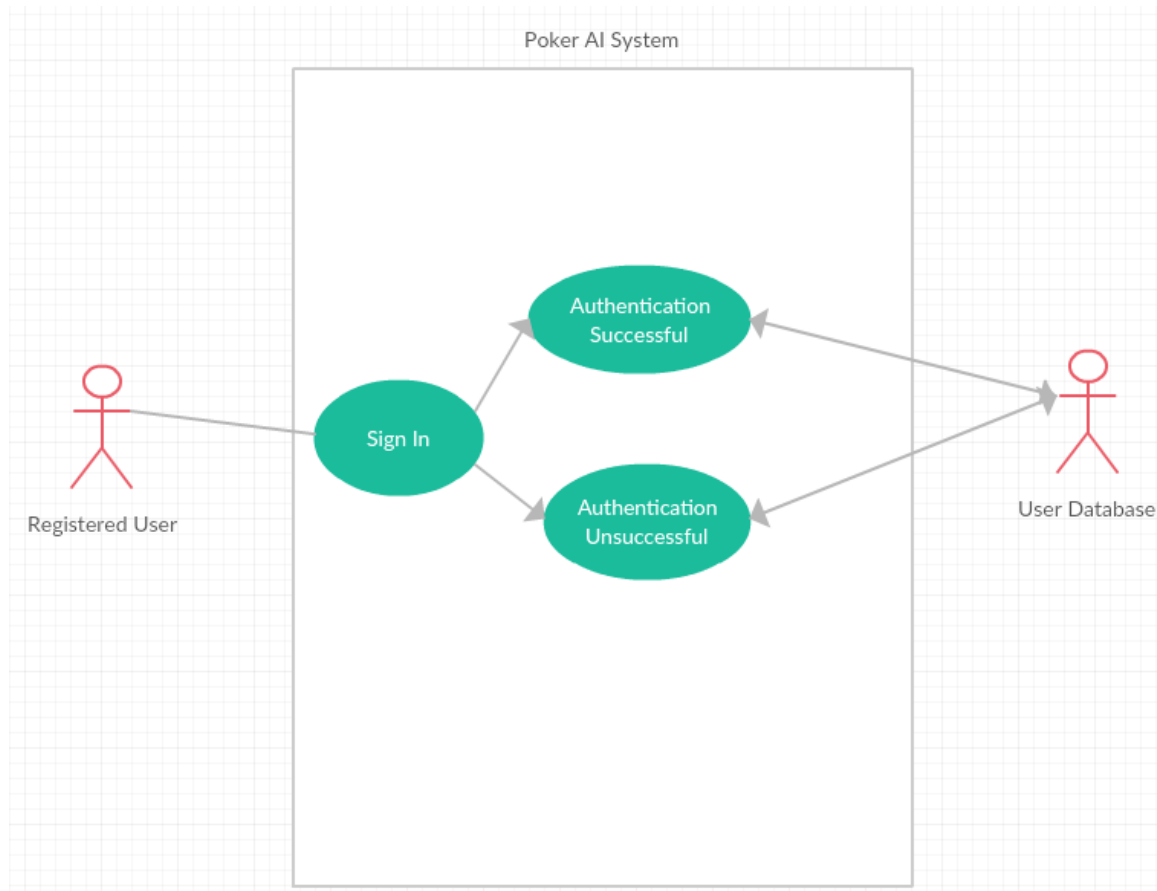
- User must attempt to register again

1.6.4 Requirement 2 <User Login>

1.6.4.1 Description & Priority

This use case describes the “Registered User” logging into the system. This requirement is key to the system in terms of allowing the user access to functionality as well as their own profile.

1.6.4.2 Use Case



Scope/Description

The scope of this use case is to log an existing user into the application, gaining access to the functionality and their own profile.

Flow Description

Precondition

User holds a valid account but has not yet authenticated onto the application during their session

Activation

This use case starts when a “Registered User” enters their credentials and presses the “Sign in” button

Main flow

- “Registered User” enters their correct credentials and presses the “Sign in” button
- System validates their credentials and provides the user access to the application
- User can now access their profile and play against the poker agent

Alternate flow

- “Registered User” enters invalid credentials and presses the “Sign in” button
- Application displays an error message, stating the user needs to re-enter their credentials and attempts to authenticate again

Termination

Main Flow

- Credentials authenticated

Alternate flow

- Credentials unauthenticated

Post condition

Main Flow

- User can access application and functionality

Alternate flow

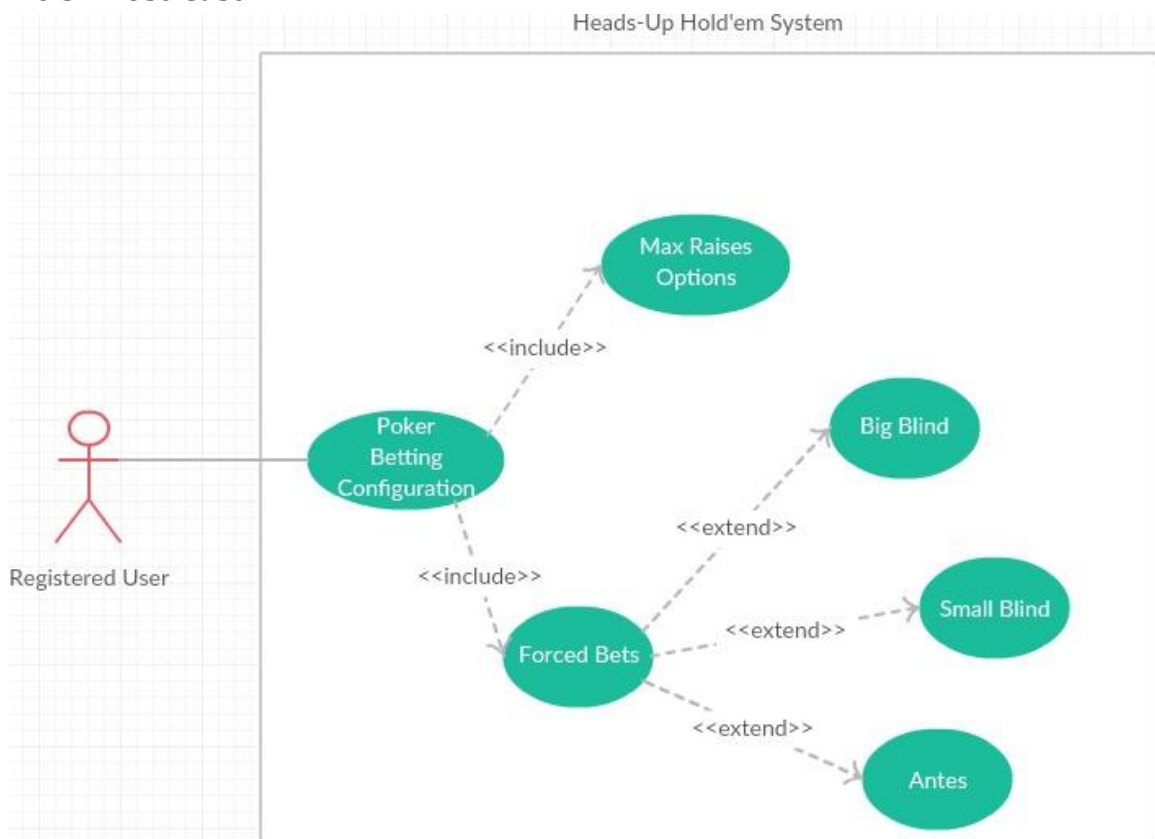
- User returned to the sign in screen

1.6.5 Requirement 3 <Betting Poker Configuration>

1.6.5.1 Description & Priority

This requirement presents the ability for the 'Registered User' to view and change the betting value options of the poker game, this includes the amount of raises and the value of blinds and antes

1.6.5.2 Use Case



Scope

The scope of this use case is to allow an existing user to alter the playstyle of their game by adjusting how many raises are available and how big the blinds consist of

Description

This use case describes the “Registered User” can adjust

Flow Description

Precondition

User holds a registered account, i.e. the account is validated through the system

Activation

This use case starts when a “Registered User” presses the “Appearance” button

Main flow

- “Registered User” presses the “Appearance” button
- User is presented a multiple of front and back card styles
- User is presented with a choice of table background colour

Alternate flow

- “Registered User” enters invalid credentials and presses the “Sign in” button
- Application displays an error message, stating the user needs to re-enter their credentials and attempts to authenticate again

Termination

Main Flow

- User navigates back to the main menu

Alternate flow

- User navigates back to the main menu

Post condition

Main Flow

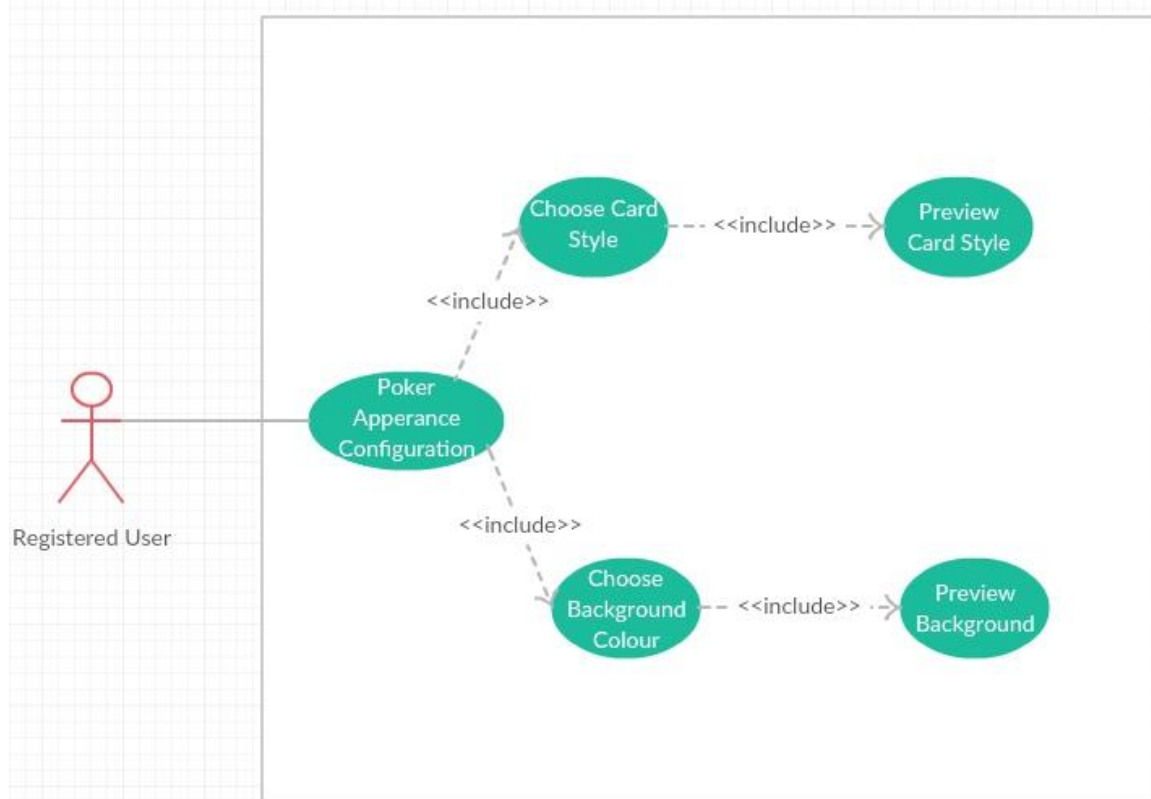
Alternate flow

1.6.6 Requirement 3 <Appearance Poker Configuration>

1.6.6.1 Description & Priority

This requirement presents the ability for the 'Registered User' to view and change the appearance of their playing cards and background table.

1.6.6.2 Use Case



Scope

The scope of this use case is to allow an existing user to change the appearance of their game before playing

Description

This use case describes the "Registered User" changing the appearance style of their cards and background.

Flow Description

Precondition

User holds a registered account, i.e. the account is validated through the system

Activation

This use case starts when a “Registered User” presses the “Appearance” button

Main flow

- “Registered User” presses the “Appearance” button
- User is presented a multiple of front and back card styles
- User is presented with a choice of table background colour

Alternate flow

- “Registered User” enters invalid credentials and presses the “Sign in” button
- Application displays an error message, stating the user needs to re-enter their credentials and attempts to authenticate again

Termination

Main Flow

- User navigates back to the main menu

Alternate flow

- User navigates back to the main menu

Post condition

Main Flow

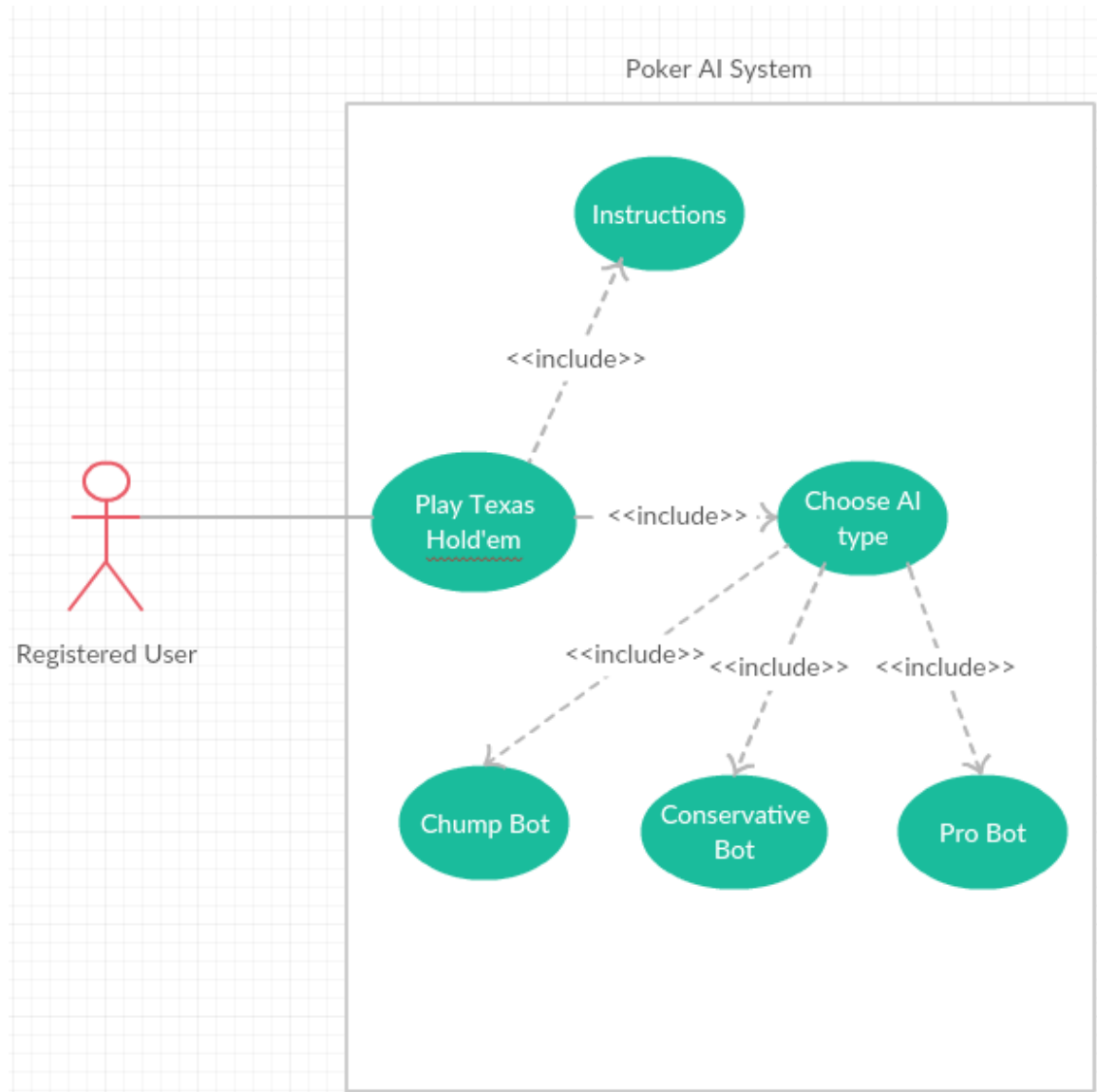
Alternate flow

1.6.7 Requirement 3 <Play Poker Menu>

1.6.7.1 Description & Priority

This requirement relates to the user accessing the core aspect of the application, playing Texas Hold'em, the user is presented with either learning the rules of Texas Hold'em or getting the option to choose three different AI types to play against.

1.6.7.2 Use Case



Scope

The scope of this use case is to allow the user to learn about Texas Hold'em if they are new, then choose to play versus an AI agent type.

Description

This use case describes the “Registered User” accessing the Play Texas Hold’em section of the application.

Flow Description

Precondition

User holds a registered account, i.e. the account is validated through the system

Activation

This use case starts when a “Registered User” presses the “Play Texas Hold’em” button

Main flow

- “Registered User” presses the “Play Texas Hold’em” button
- User is presented with Instructions menu or Choose AI type

Alternate flow

- User remains on the original page

Termination

Main Flow

- User navigates back to the main menu

Alternate flow

- User navigates back to the main menu

Post condition

Main Flow

- User can access application and functionality

Alternate flow

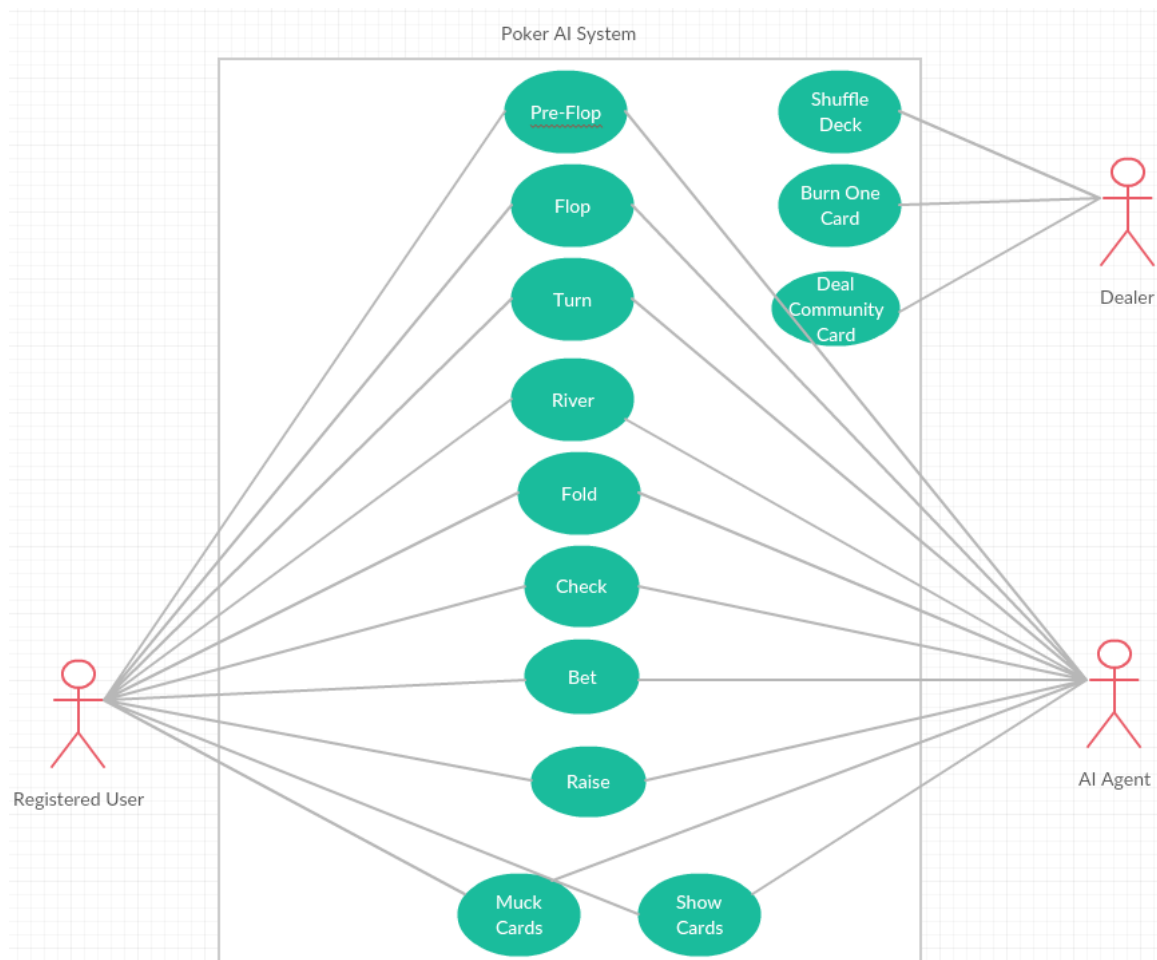
- User returned to the sign in screen

1.6.8 Requirement 3 <Play Versus AI>

1.6.8.1 Description & Priority

This requirement is the core aspect of the application. The “Registered User” plays Texas Hold’em against the AI agent type of their choice.

1.6.8.2 Use Case



Scope/Description

The scope of this use case is the “Registered User” plays Texas Hold’em against the AI agent type of their choice.

Flow Description

Precondition

User holds a registered account, i.e. the account is validated through the system and has chosen the AI agent type they wish to play against

Activation

This use case starts when a “Registered User” chooses the AI agent type they wish to play against

Main flow

- “Registered User” choose the AI agent type they wish to play against
- User is put into a heads up or one on one Texas Hold'em game against the AI agent they chose

Alternate flow

- User remains on the “Play Poker Menu”

Termination

Main Flow

- User Win's versus the AI agent by taking all the agent's chips
- User Loses versus the AI agent by losing all their chips to the agent

Alternate flow

- User navigates back to the “Play Poker Menu”

Post condition

Main Flow

Alternate flow

1.7 User Requirements

The user requirement of Poker AI is to incorporate an application to allow Poker and AI enthusiasts to enjoy fun and simple Poker.

Internet access: The device will need Internet access to use the application because it must connect to a server to retrieve information. The faster the internet the faster the server get and post requests **Invalid source specified..**

1.7.1 User requirements

This section describes the set of objectives and requirements for the system from the customer's perspective. What are the clients saying they want?

- A new user must be able to sign up to the system
- An already registered user must be able to log into the system
- A user should be able to check the rules on Texas Hold'em through the application
- Should be able to play Texas Hold'em
- A user should be able refill their play money chips that is capped to a certain amount every few hours
- A user should be able to purchase play money chips if they so choose to
- A user should be able to view the leader boards through the application

1.7.2 Environmental requirements

These are the vital requirements that must be present when developing the application.

- Internet Access: Internet access is required to test functions in application and connecting to database.
- Laptop(Window): This application will be developed Windows laptop with android studio as the Android development IDE.

- Photoshop/Paint: Photoshop was used to customize any images and graphical assets used during the development of my application.

1.7.3 Usability requirements

This requirement will cover and evaluate the usability requirements for the system application. This outline the standards and objectives to be met regarding the systems.

- **Ease of use:** The application must be user friendly and easy to use.
- **Understandability:** The system should be understandable to the use. Easy to follow the functionates.
- **Operability:** The system should perform as mentioned in the requirement. The app should be consistent in terms of functionality.
- **Attractiveness:** The application should be appealing to users (GUI, Design and layout). The app should use colours that is easy to for the eye.

1.8 *Non-Functional Requirements*

The divergence between Functional Requirement and Non-Functional Requirement is Functional Requirement deal with what the system shall and much do, while Non-Functional Requirement focuses on, How the system operate.

1.8.1 Performance/Response time requirement

The AI agent should play their turn in a timely manner, to maintain the flow of the game although play in an efficient way to not reveal information. Taking your time in Texas Hold'em is a key aspect of the game to not reveal information to your

opponent, although if the AI takes too long to act every hand, the user will get frustrated and possibly stop playing.

1.8.2 Availability requirement

The application should be available to be used at any given time. Once a user has accessed the application it should be fully functional and accessible to the user. The application should be free from downtime and if any bugs or errors appear they must immediately be repaired or removed to insure the application services remain up and running to the users.

1.8.3 Data requirements

User's data is stored within a MySQL database. This data will be secure and encrypted using an encryption library. As this application will be free and include no monetary aspects, this reduces the impact of storing sensitive payment

1.8.4 Security requirement

The application should include a robust level of security that ensures users personal information they provided is secured and encrypted on the server. The application files must be secure to avoid any attacks or information leakage. The application should be designed to only accept passwords that contain the strong password criteria.

1.8.5 Reliability requirement

In many systems reliabilities are a big consideration during the development stage. If a system keeps crashing or has a lot of software bugs this will affect the overall reliability of the application and affect the user's use of the application services. I will take a range of measures into account ensuring that all software bugs have be eliminated through varies testing methods such as verification, validation, integration testing, functional testing, system testing and. Logical errors will be removed where possible. The system should be able to cope with minor issues that may arise because of internal factors therefore making it reliable.

1.8.6 Maintainability requirement

The application should maintain updated, with the leader boards showcasing the correct information in real time. The Poker AI's will be constantly reviewed to assess any weaknesses they may have and address them. This will allow for future improvement of the AI's as time goes on.

From a security prospective there are many black hat hackers in the current time. One of the goal is to develop an application that not only completes its intended functionality with ease but also impresses and exceeds users' expectations. Where the application happens to fall in terms of performance, functionality or it needs to be updated, several key aspects will be taking to help maintain the application and to help security:

- Defects/Vulnerability
This involves as a developer that reviewing the area of concern for possible abnormalities and addressing them appropriately.
- Code Quality
Looking over the existing code where there lies a potential problem and either patch or improve it.
- Reducing Redundancy
To make the application easy to understand and to make sure that two pieces of code are not doing the same thing, the aim is to eliminate redundancy altogether, assisting in maintaining the steadiness of the overall system.

1.8.7 Portability requirement

The application and website must be accessible from any device which can use an internet browser. The application must work as smooth as possible across all modern browsers such as Edge, Safari, Chrome and Firefox.

1.8.8 Extendibility requirement

The system shall be extensible enough so that during future development there can added additional functionalities such as various types of AI agents or expand further on the art style of the application

1.8.9 Recover requirement:

The recovery is the area of security that have to be put in place if there were a significant negative events.

In the case of system shutdown and no response to the user. The system should be down and shouldn't take no more than one to two working day for the system to be back. Also, Information will be send to the user to let them know when the system is back working.

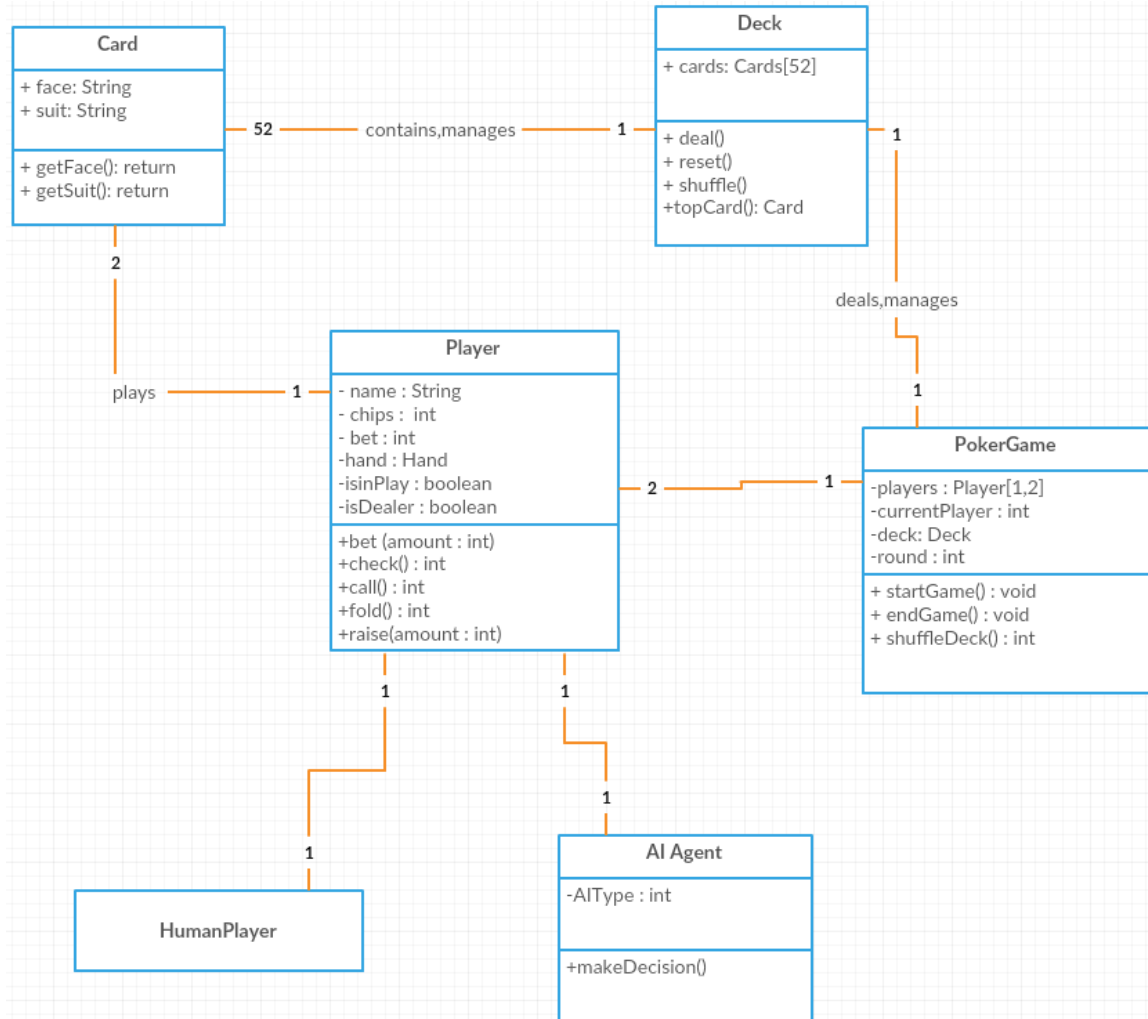
1.9.0 Reusability requirement:

Reusability is a valid requirement for all projects that involve software development. The application will aim to have multiple functions and other code that can be re-used and implemented into further or new developments going forward or indeed when creating something new.

Reusability is an important factor and requirement as applications in the real world are based hugely on re-used code. The goal is to develop unique code that can be understand and that am comfortable implementing into new environments.

The login/registration system is a prime example of this, the vast majority of mobile application with user interaction require user login. The database which will be implemented and connected should be flexible and reusable with other applications where some type of information is being stored.

1.9 Design and Architecture



1.10 Implementation

StandardHand

StandardHand, checks all available hands that can be made during a poker game ranging from a StraightFlush to High Card. The decision to leave out a royal flush is to confine the decision making of the AI to a more strict bases revolving around the hands that are more likely to happen. The probability of a royal flush is 0.000154% or 649,739:1. Due to the unlikeliness of achieving the hand, I felt it was more beneficial for the AI to not have it within it's hand range and focus on achieving hands which are more likely to occur.

```
public StandardHand(final Deck cards) {
    this.cards = cards;

    if (isStraightFlush(cards)) {
        bestRank = StandardHand.Ranking.STRAIGHT_FLUSH;
    } else if (isFourOfAKind(cards)) {
        bestRank = StandardHand.Ranking.FOUR_OF_A_KIND;
    } else if (isFullHouse(cards)) {
        bestRank = StandardHand.Ranking.FULL_HOUSE;
    } else if (isFlush(cards)) {
        bestRank = StandardHand.Ranking.FLUSH;
    } else if (isStraight(cards)) {
        bestRank = StandardHand.Ranking.STRAIGHT;
    } else if (isThreeOfAKind(cards)) {
        bestRank = StandardHand.Ranking.THREE_OF_A_KIND;
    } else if (isTwoPairs(cards)) {
        bestRank = StandardHand.Ranking.TWO_PAIR;
    } else if (isOnePair(cards)) {
        bestRank = StandardHand.Ranking.ONE_PAIR;
    } else {
        high = null;
        low = null;
        usesLow = false;
        usesAllCards = false;
        kickers = new Deck(cards);
        kickers.limitTo(5);

        bestRank = StandardHand.Ranking.NO_PAIR;
    }
}
```

Figure 4: Hand Ranks

HandDetails

HandDetails makes boolean checks to see if a hand from StandardHand has been made, such as a Flush. The isFlush method showcases that a boolean check is being made within the HandDetails class, to ensure that a player must make use of all of their cards and must be equal to 5 aswell as be of the same suit.

```
protected boolean isFlush(final Deck deck) {
    usesLow = false;
    usesAllCards = true;

    for (Suit suit : Suit.values()) {
        Deck myDeck = deck.getSuit(suit);

        if (myDeck.size() >= 5) {
            Collections.sort(myDeck);

            high = myDeck.get(0).getRank();
            kickers.clear();

            myDeck.limitTo(5);
            deck.clear();
            deck.addAll(myDeck);

            return true;
        }
    }
}
```

Figure 5: isFlush Method

Card

Represents a play card from a set of cards {0, 51} which map to cards having a suit {0, 3} clubs, diamonds, hearts, spades and a face value {0, 12} 2, Ace

```
public Card(final Suit suit, final Rank rank) {
    this.suit = suit;
    this.rank = rank;
}
```

Figure 6: Card method

Rank

Defines the standard ranks in a pack of cards.

Deck

A deck of 52 Cards which can be dealt and shuffled

DealCard

Deals a single card to each active player, starting with the specified player and working clockwise.

The boolean “isPublic” checks to see whether the card is a public card or not

```
protected void dealCard(final Player start, final boolean isPublic) {
    int i = players.indexOf(start);
    for (int x = 0; x < numplayers; x++) {
        final Player player = players.get((i + x) % numplayers);

        if (!player.isOut() && !player.hasFolded()) {
            final Card card = deck.deal();
            card.setPublic(isPublic);

            player.dealCard(card);
            notifyCardDealt(player, card);
            notifyPlayerCardsUpdated();
        }
    }
}
```

Player

This class creates instances of the player which enables quicker access and efficient storage of the information, regardless of which player it is for. The PokerGame class creates new instances of player at every new iteration. The player instance stores information such as history and actions.

Calls on the functions doOpenCheck and doCallRaiseFold.

- doCallRaiseFold
 - Determines whether the AI can call, raise or fold
- doOpenCheck
 - Determines whether the AI will open or check

```

public Player(final GameInterface game, final String name, final int cash, final PlayerController controller) {
    this.name = name;
    this.cash = cash;
    this.game = game;

    this.controller = controller;
    controller.setPlayer(this);
    controller.setGame(game);
}

public Deck getCards() {
    return cards;
}

public Deck getBestDeck() {
    calculateBestDeck();

    return bestDeck;
}

public void calculateBestDeck() {
    bestDeck = game.getBestDeck(cards);
}

public OpenCheck doOpenCheck() {
    OpenCheck res = controller.doOpenCheck();

    if (cash <= 0 && res == OpenCheck.OPEN) {
        res = OpenCheck.CHECK;
    }

    return res;
}

public CallRaiseFold doCallRaiseFold(final int callAmount, boolean canRaise) {
    CallRaiseFold res = controller.doCallRaiseFold(callAmount, canRaise);

    if (cash - callAmount <= 0 && res == CallRaiseFold.RAISE) {
        res = Player.CallRaiseFold.CALL;
    }

    return res;
}

```

Figure 7: Player class

Suit

Defines the four suits contained within a deck of playing cards

```

public enum Suit {

    CLUBS,
    SPADES,
    DIAMONDS,
    HEARTS;

}

```

Figure 8: Suit class

ChumpBot

This is the initial AI developed for testing the application and GUI is ensure everything is fully functional. ChumpBot consists of random moves with no decision making based on a hand evaluation.

```
public class ChumpBot implements PlayerController {

    /** The player that we're controlling. */
    protected Player player;

    /** The game that we're playing. */
    protected GameInterface game;

    @Override
    public CallRaiseFold doCallRaiseFold(final int callAmount, final boolean canRaise) {
        if (Math.random() < 0.3) {
            if (player.getCash() - callAmount < game.getBigBlind() || !canRaise) {
                return CallRaiseFold.FOLD;
            } else {
                return CallRaiseFold.RAISE;
            }
        } else {
            return CallRaiseFold.CALL;
        }
    }

    @Override
    public OpenCheck doOpenCheck() {
        if (Math.random() < 0.5 || player.getCash() < game.getBigBlind()) {
            return OpenCheck.CHECK;
        } else {
            return OpenCheck.OPEN;
        }
    }
}
```

Figure 9: ChumpBot class

Conservative Bot

The conservative AI extends from the ChumpBot, the decision making ability prior to the flop is based on ranking both cards in the agents starting hand. The AI agent compares it's starting hand based on their rank and decides whether to call, raise or fold.

Post starting hand, the conservative agent will act in the same manner as the Chump agent, making randomly based decisions with no information of community cards taking impact.

```
public class ConservativeBot extends ChumpBot {  
    @Override  
    public CallRaiseFold doCallRaiseFold(int callAmount, boolean canRaise) {  
        Card c1 = player.getCards().get(0);  
        Card c2 = player.getCards().get(1);  
  
        if (c1.getSuit() == c2.getSuit() || Math.abs(c1.getRank().compareTo(c2.getRank())) < 2) {  
            return super.doCallRaiseFold(callAmount, canRaise);  
        } else {  
            return CallRaiseFold.FOLD;  
        }  
    }  
}
```

Figure 9: ConservativeBot Class

ProBot

The ProBot agent extends from the Conservative agent, although the contrast between the two is present during the flop, turn and river. As the ProBot makes use of HashMap values, that allows for logical decision making based on the agent's current hand based on the assigned values.


```

public ProBot(final GameInterface game) {
    game.registerObserver(this);

    handRanks.put(StandardHand.Ranking.FLUSH, 4047644);
    handRanks.put(StandardHand.Ranking.FOUR_OF_A_KIND, 224848);
    handRanks.put(StandardHand.Ranking.FULL_HOUSE, 3473184);
    handRanks.put(StandardHand.Ranking.NO_PAIR, 23294460);
    handRanks.put(StandardHand.Ranking.ONE_PAIR, 58627800);
    handRanks.put(StandardHand.Ranking.STRAIGHT, 6180020);
    handRanks.put(StandardHand.Ranking.STRAIGHT_FLUSH, 41584);
    handRanks.put(StandardHand.Ranking.THREE_OF_A_KIND, 6461620);
    handRanks.put(StandardHand.Ranking.TWO_PAIR, 31433400);
}

```

Figure 10: ProBot

HashMap values

The ProBot agent also can make bluffs based on it's current hand and chip amount.

- callRaiseFoldBluff
 - Once the ProBot has reached endGame, which means all community cards have been dealt. It can make a bluff depending on if the agent is able to raise

```

if (endGame) {
    if (canRaise && (shouldBluff || (!shouldFold && Math.random() > 0.5))) {
        return CallRaiseFold.RAISE;
    } else if (shouldFold) {
        return CallRaiseFold.FOLD;
    } else {
        return CallRaiseFold.CALL;
    }
} else {
    return super.doCallRaiseFold(callAmount, canRaise);
}

```

Figure 11: callRaiseFoldBluff Method

- getRaiseBluff
- When bluffing the agent will take into account it's cash stack and make a raise accordii

```

if (endGame) {
    if (shouldBluff) {
        return minimum + (int) (Math.random() * (player.getCash() - minimum)/2);
    } else {
        return minimum + (int) (Math.random() * (player.getCash() - minimum));
    }
} else {
    return super.getRaise(minimum);
}

```

Figure 12: getRaiseBluff Method

Hand Details

The HandDetails abstract class contains boolean checks for all available hands in poker. These methods range from One Pair, to a Straight Flush, the class also contains relevant checks for kickers and high/low cards.

- The isFlush method within the HandDetails class, is a simple check for all cards must be used and that the 5 cards must be of all relevant suit
- The method will dismiss kickers in this case, as it requires 5 cards to make a flush

```
protected boolean isFlush(final Deck deck) {  
    usesLow = false;  
    usesAllCards = true;  
  
    for (Suit suit : Suit.values()) {  
        Deck myDeck = deck.getSuit(suit);  
  
        if (myDeck.size() >= 5) {  
            Collections.sort(myDeck);  
  
            high = myDeck.get(0).getRank();  
            kickers.clear();  
  
            myDeck.limitTo(5);  
            deck.clear();  
            deck.addAll(myDeck);  
  
            return true;  
        }  
    }  
}
```

Figure 13: isFlush Method

- The isThreeOfAKind method is another example within the HandDetails class,
- Checks for the highest ranking 3 cards of the same value
- Kickers are used in this case but limited to 2


```

protected boolean isThreeOfAKind(final Deck deck) {
    usesLow = false;
    usesAllCards = false;

    for (Rank rank : Rank.values()) {
        if (deck.getRank(rank).size() == 3) {
            high = rank;
            kickers = new Deck(deck);
            kickers.removeByRank(high, 3);
            kickers.limitTo(2);
            return true;
        }
    }
}

```

Figure 14: isThreeOfAKind Method

Game Details

In comparison to the Hand Details abstract class, the Game Details class contains all relevant information in regards to how an Heads-Up No Limit Poker game operates.

This is limited too implementations such as , the number of players and their specified properties, such as the player is currently still in play, if they have folded or if they are all in. The size of both the big and small blind, the antes, the number of raises that are allowed, as well as keeping track of the cards that have been dealt to ensure they won't be duplicated.

```

public int countPlayers(final boolean mustBeIn,
    final boolean mustNotFolded, final boolean mustNotAllIn) {
    int count = 0;

    for (Player player : players) {
        if ((!mustBeIn || !player.isOut())
            && (!mustNotFolded || !player.hasFolded())
            && (!mustNotAllIn || !player.isAllIn())) {
            count++;
        }
    }

    return count;
}

```

Figure 15: countPlayers Method

```
protected void doSmallBlind(final Player player) {
    player.forceBet(bigblind / 2);

    notifyPlaceBlind(player, bigblind / 2, "small blind");
}

protected void doBigBlind(final Player player) {
    player.forceBet(bigblind);

    notifyPlaceBlind(player, bigblind, "big blind");
}
```

Figure 16: doSmallBlind/doBigBlind Methods

The GameDetails class also contains the method for dealing cards to each player. The variable start, deals a single card to the specific player working clockwise. The isPublic variable is a boolean check to see whether the card is a public card or not.

```
protected void dealCard(final Player start, final boolean isPublic) {
    int i = players.indexOf(start);
    for (int x = 0; x < numplayers; x++) {
        final Player player = players.get((i + x) % numplayers);

        if (!player.isOut() && !player.hasFolded()) {
            final Card card = deck.deal();
            card.setPublic(isPublic);

            player.dealCard(card);
            notifyCardDealt(player, card);
            notifyPlayerCardsUpdated();
        }
    }
}
```

Figure 17: dealCard Method

SaveSettings

After the user chooses what settings they which to play with, such as their chip amount, forced bets and aesthetics of their game. The application will save their settings to a config file using the saveSettings method.

```
public void saveSettings(final String name) {
    try {
        final ObjectOutputStream oos
            = new ObjectOutputStream(new FileOutputStream(
                new File(getConfigDir(), name + ".config")));
        oos.writeObject(new ConfigVersioner());
        oos.writeObject(game);
        oos.writeObject(playerPanel.getData());
        oos.writeObject(bettingPanel.getData());
        oos.writeObject(appearancePanel.getData());
    } catch (IOException ex) {
        System.err.println("Unable to save settings to " + name);
        ex.printStackTrace();
    }
}
```

LoadSettings

The loadSettings method will grab the information from the saved settings config file after the application is closed, and will load their exact previous settings.

```
public void loadSettings(final String name) {
    try {
        final ObjectInputStream ois
            = new ObjectInputStream(new FileInputStream(
                new File(getConfigDir(), name + ".config")));

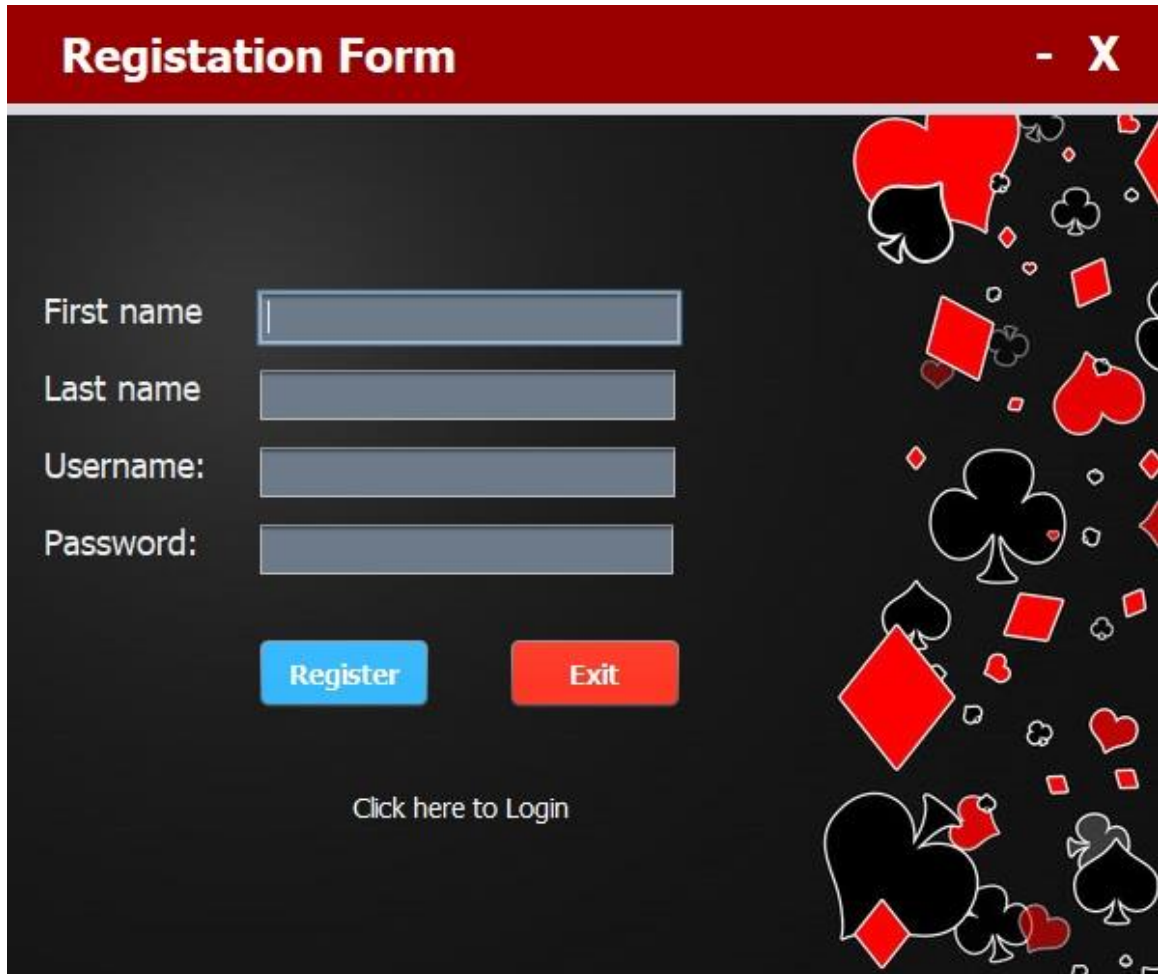
        ois.readObject();

        final GameInfo gameInfo = (GameInfo) ois.readObject();
        final List<Object[]> playerData = (List<Object[]>) ois.readObject();
        final Object[] bettingData = (Object[]) ois.readObject();
        final Object[] styleData = (Object[]) ois.readObject();
    }
}
```

1.11 Graphical User Interface (GUI) Layout

Provide screenshots of key screens and explain.

Registration Form

A screenshot of a registration form window. The window has a dark red title bar with the text "Registration Form" and a close button "X". The main area has a black background with a pattern of red and white playing cards (hearts, spades, diamonds, clubs) on the right side. On the left, there are four input fields labeled "First name", "Last name", "Username:", and "Password:". Below these fields are two buttons: a blue "Register" button and a red "Exit" button. At the bottom center, there is a link that says "Click here to Login".

This design allows the user to create an account for identity authentication

It takes the following input:

- Username as a unique identifier to recognise the user
- Full Name of the User
- Password to authorise the user

Security

The registration form attempts to register a user's account to the database and hashes the user's password within the database using an MD5 Hash

```
public static String md5(String msg) {
    try {
        MessageDigest md = MessageDigest.getInstance("MD5");
        md.update(msg.getBytes());
        byte byteData[] = md.digest();
        //convert the byte to hex format method 1
        StringBuffer sb = new StringBuffer();
        for (int i = 0; i < byteData.length; i++) {
            sb.append(Integer.toString((byteData[i] & 0xff) + 0x100, 16).substring(1));
        }
        return sb.toString();
    } catch (Exception ex) {
        return "";
    }
}
```

Figure 18: Database Hash

SELECT * FROM `register`

☐ Show all | Number of rows: 25 | Filter rows: Search this table

+ Options

	ID	FirstName	SurName	UserName	Password
<input type="checkbox"/> Edit <input type="checkbox"/> Copy <input type="checkbox"/> Delete	14	TestUser	Murray	Lee	5f4dcc3b5aa765d61d8327deb882cf99

☐ Check all | With selected: ☐ Edit ☐ Copy ☐ Delete ☐ Export

Figure 19: User's information logged

The registration forum also contains a log file that will handle any form of invalid login entries which includes a timestamp of the error.

```

public class Log {

    public static void main(String[] args) throws IOException {

        Logger logger = Logger.getLogger("MyLog");
        FileHandler fh;

        try {

            // This block configure the logger with handler and formatter
            fh = new FileHandler("C:/Users/lee-e/Documents/NetBeansProjects/SecureProjectMyLogFile.log");

            logger.addHandler(fh);
            SimpleFormatter formatter = new SimpleFormatter();
            fh.setFormatter(formatter);

            // the following statement is used to log any messages
            logger.info("Log Attempt");

        } catch (SecurityException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }

        logger.info("Test");

    }

}

```

Figure 19: Log File


Login Form

Login Form - X

Username:

Password:

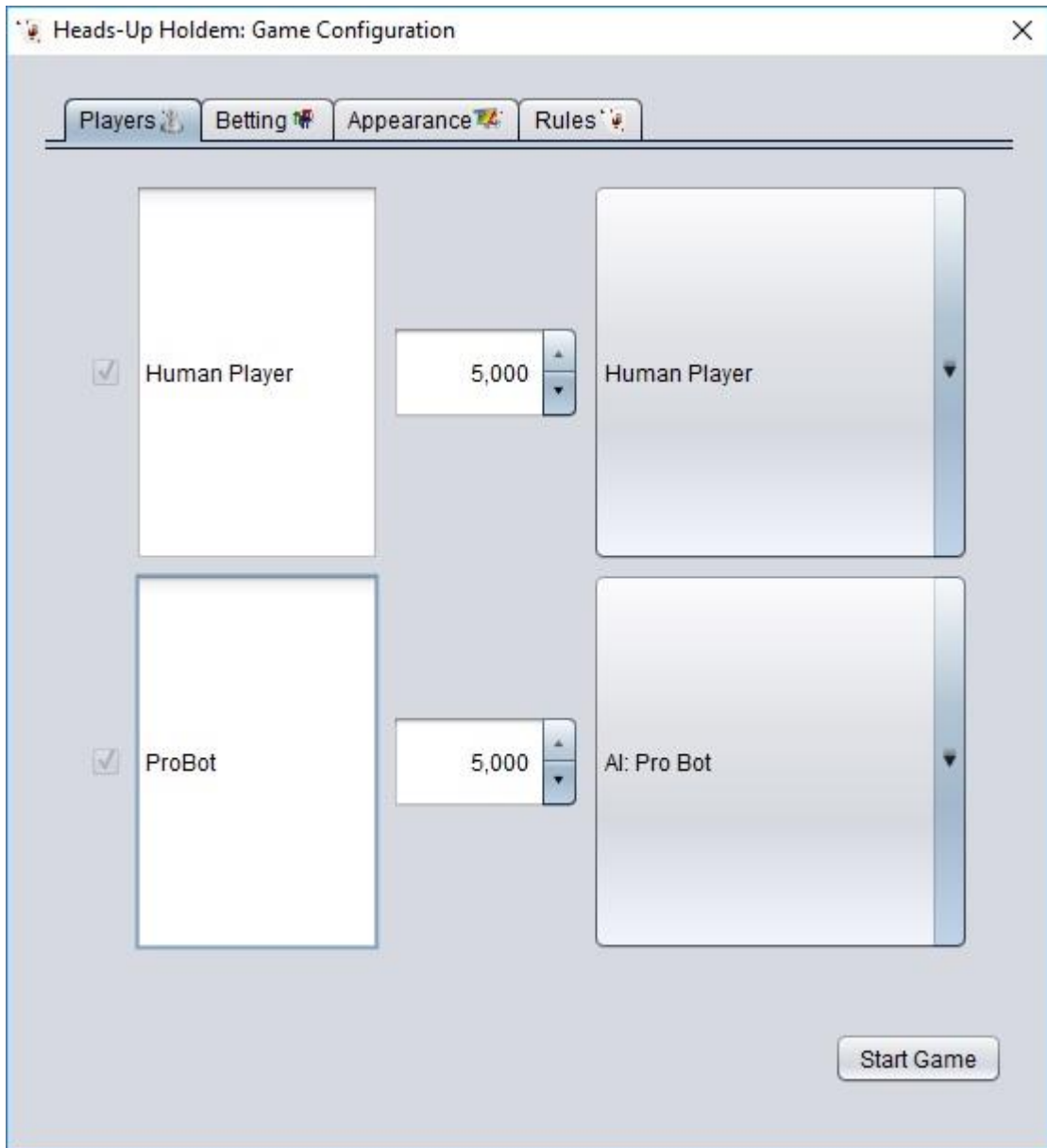
[click here to create a new account](#)



Poker Player Configuration

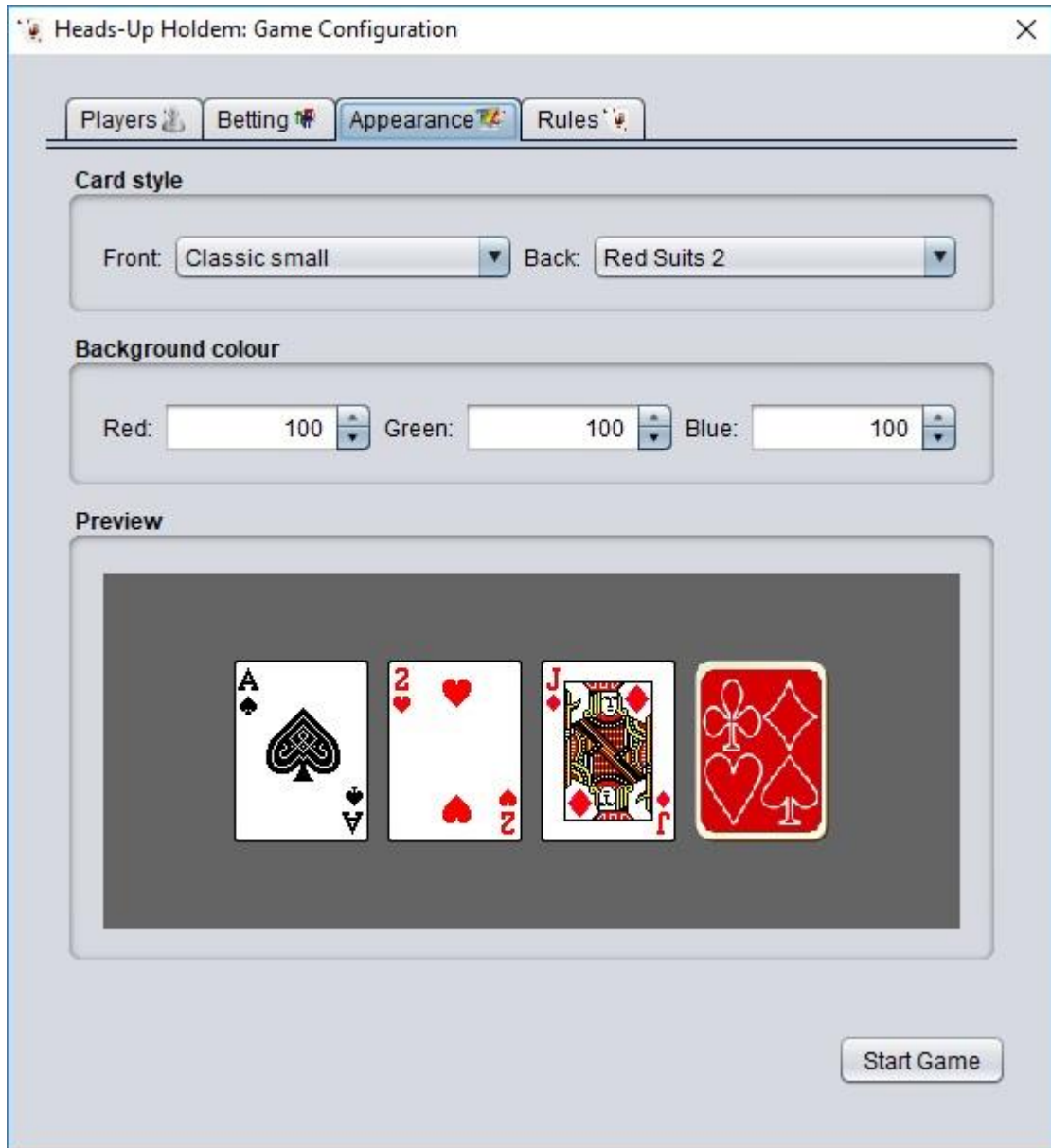
The Heads-Up Hold'em player configuration menu allows the user to choose what type of AI agent they wish to play against, which include the ChumpBot, ConservativeBot and the ProBot.

The player also has the ability to alter the amount of chips counts of each players, this allows for the type of pace that the player can play at.



Appearance Configuration Menu

The appearance configuration menu offers the user to alter the style of their playing cards and background colour of the table.



Betting Configuration Menu

The betting configuration allows the user to change the number raises allowed during each hand and forced bets amounts paid before starting their game.

The screenshot shows a window titled "Heads-Up Holdem: Game Configuration" with a close button (X) in the top right corner. Below the title bar is a tabbed interface with four tabs: "Players", "Betting", "Appearance", and "Rules". The "Betting" tab is currently selected. The main area is divided into two sections: "Options" and "Forced bets".

Options

Max raises:

Forced bets

Big blind:

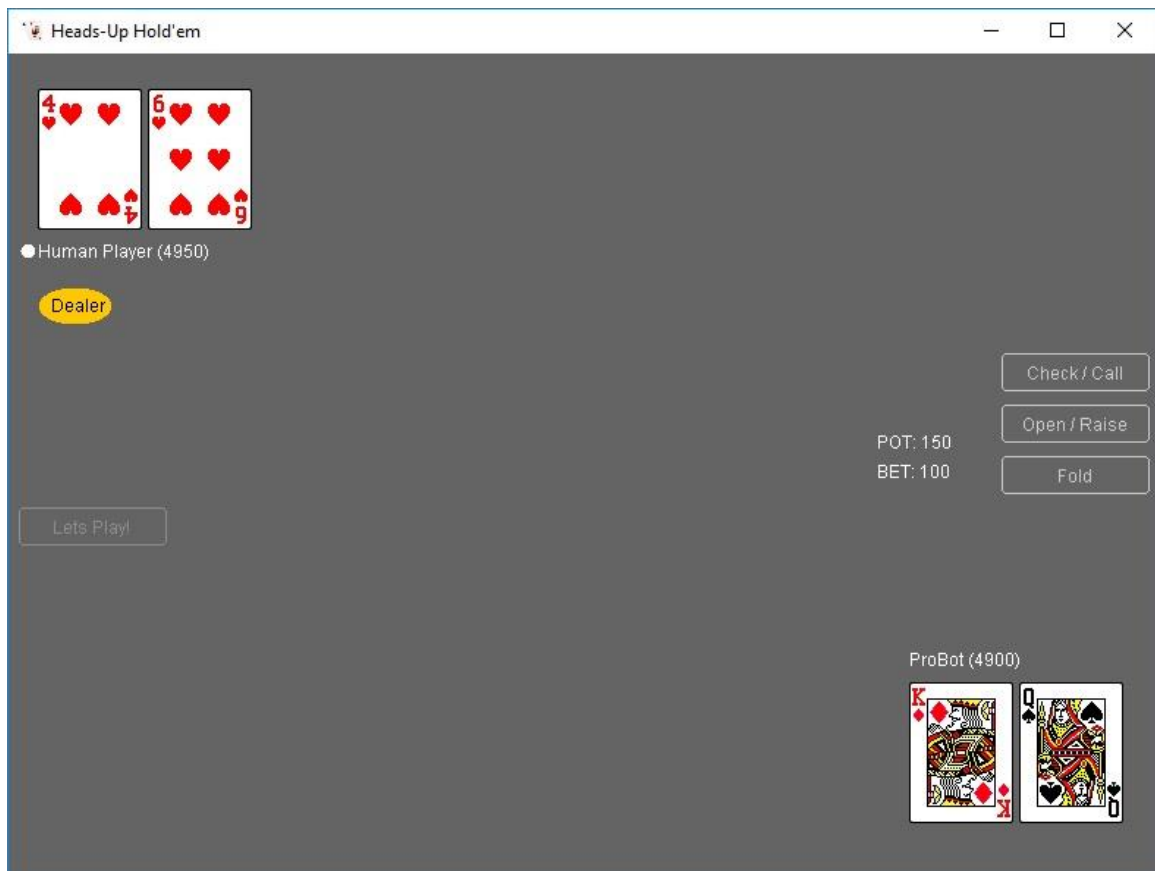
Small blind:

Ante:

At the bottom right of the window is a "Start Game" button.

Poker Table

This design is the main aspect of the application, this showcases the user playing No-Limit Texas Hold'em versus the AI agent. The user can begin the game by clicking the "Let's Play!" button, this will shuffle the deck and give both players two cards.



1.12 Testing

Testing

Since the application will be heavily fragmented, it will be needed to constantly be tested to ensure the algorithm is working correctly.

Unit Testing

Manual testing will be done to ensure the calculations are correct, i.e they give the predicted results.

All created objects will be tested to ensure that the expected data is assigned to them and consequently used in game class. The card shuffling which forms a crucial aspect of the system will need to be exhaustively tested.

The program will also need to be tested to ensure the randomness of its actions are maintained.

User Testing

This method of testing will be used during every stage of the development. During User testing, information such as the player's cards or amount of chips they hold will need to be correct.

The program's actions must be heavily tested to prevent as little as much bugs as appearing, such as a user may be able to press the bet button twice during their hand. This could cause unexpected and unstable behavior in the application and could never be detected without user testing.

Artificial Intelligence Testing

Once an environment for the agents to act in was established, experiments provide the simplest way to determine the effectiveness of changes, or to identify potential problems and shortcomings of the agents. The primary method, although difficult to interpret but presented as anecdotal evidence, involved playing agents playing against one another.

The self-play simulation offers a convenient method for the comparison of various versions of the agent. One simple application would be to play five 10 hands with a newer version of the agent against 10 hands of an older version of the agent. The likelihood of the new components should have improved the program against itself, then the newer version will win against the older version.

This process was continued to successfully increase the correct decision making of the agent.

Over the course of self-play experiments, there were 5 variations of the ProBot agent that was tested, using different components of hand ranking values and betting strategies.

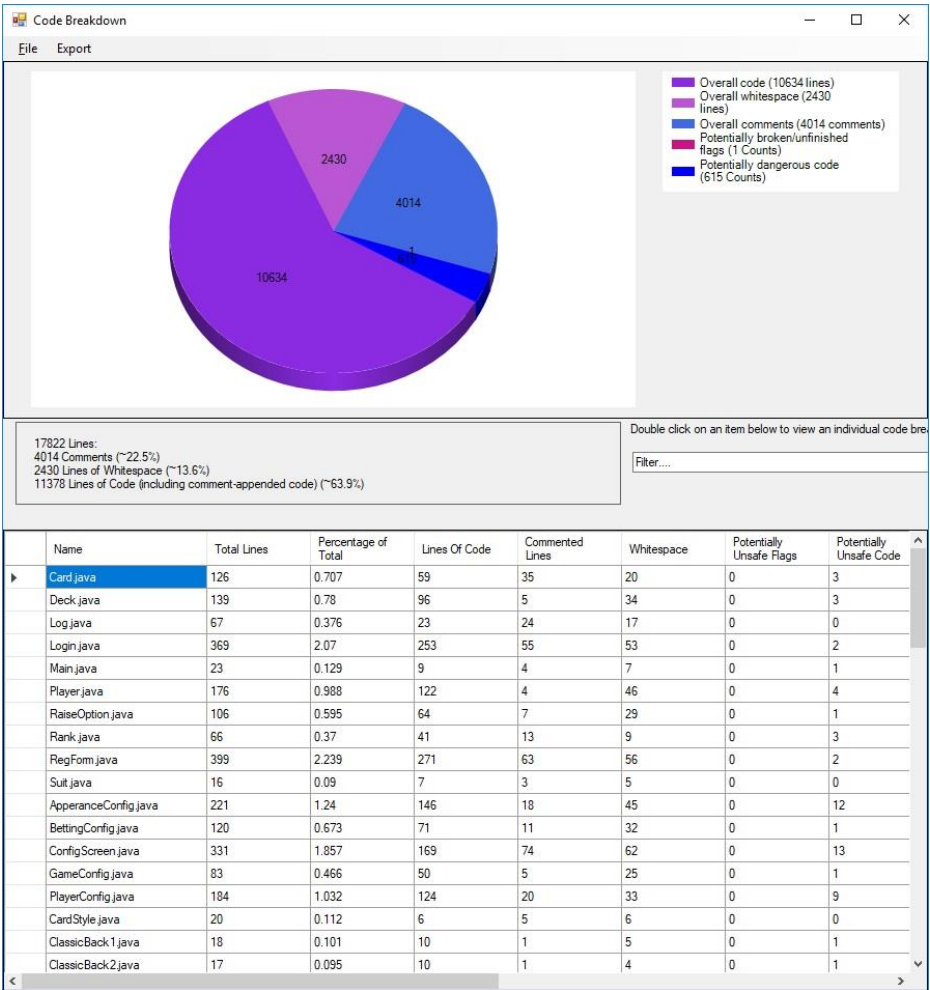
Pre-Flop Simulated Results

Pockets : AAo 85.14%	Pockets : K5s 55.55%	Pockets : 95s 45.63%
Pockets : KKo 82.22%	Pockets : K7o 55.33%	Pockets : 87o 45.45%
Pockets : QQo 79.63%	Pockets : A3o 55.28%	Pockets : J3o 45.26%
Pockets : JJo 77.2%	Pockets : K4s 54.71%	Pockets : J2s 45.09%
Pockets : TTo 74.57%	Pockets : Q7s 54.57%	Pockets : 22o 44.98%
Pockets : 99o 71.08%	Pockets : T9s 54.48%	Pockets : 85s 44.79%
Pockets : 88o 68.24%	Pockets : K6o 54.47%	Pockets : Q2o 44.79%
Pockets : AKs 67.67%	Pockets : J8s 54.45%	Pockets : 96o 44.6%
Pockets : AQs 66.41%	Pockets : K3s 54.03%	Pockets : T5o 44.11%
Pockets : AKo 66%	Pockets : Q8o 53.69%	Pockets : 75s 43.88%
Pockets : 77o 65.74%	Pockets : J9o 53.58%	Pockets : 86o 43.57%
Pockets : AJs 65.72%	Pockets : K5o 53.33%	Pockets : 94s 43.4%
Pockets : AQo 64.79%	Pockets : Q6s 53.21%	Pockets : 65s 43.35%
Pockets : ATs 64.79%	Pockets : A2o 52.89%	Pockets : T4o 43.17%
Pockets : AJo 63.85%	Pockets : J7s 52.87%	Pockets : 93s 42.86%
Pockets : KQs 63.84%	Pockets : K4o 52.65%	Pockets : 84s 42.86%
Pockets : KJs 63.22%	Pockets : 33o 52.4%	Pockets : 76o 42.69%
Pockets : A9s 63.04%	Pockets : T8s 52.34%	Pockets : 95o 42.58%
Pockets : ATo 62.73%	Pockets : Q5s 52.22%	Pockets : T3o 42.54%
Pockets : A8s 62.51%	Pockets : Q7o 51.83%	Pockets : T2s 42.48%
Pockets : 66o 62.39%	Pockets : Q4s 51.63%	Pockets : 74s 41.8%
Pockets : KTs 61.72%	Pockets : K2s 51.48%	Pockets : J2o 41.52%
Pockets : KQo 61.42%	Pockets : J8o 51.46%	Pockets : 85o 41.29%
Pockets : KJo 61.05%	Pockets : 98s 51.45%	Pockets : 54s 41.07%
Pockets : A7s 60.92%	Pockets : K3o 51.44%	Pockets : 94o 40.92%
Pockets : A9o 60.88%	Pockets : T9o 51.36%	Pockets : 75o 40.91%
Pockets : QJs 60.82%	Pockets : Q3s 51.2%	Pockets : 64s 40.9%
Pockets : KTo 60.13%	Pockets : Q6o 50.73%	Pockets : 83s 40.58%
Pockets : A8o 59.95%	Pockets : J6s 50.56%	Pockets : 92s 40.14%
Pockets : A6s 59.92%	Pockets : T7s 50.5%	Pockets : 73s 39.86%
Pockets : K9s 59.89%	Pockets : Q5o 50.1%	Pockets : 65o 39.79%
Pockets : QTs 59.7%	Pockets : J7o 50.05%	Pockets : 93o 39.71%
Pockets : A5s 59.51%	Pockets : J4s 49.42%	Pockets : 84o 39.4%
Pockets : 55o 59.15%	Pockets : T8o 49.41%	Pockets : 63s 39.15%
Pockets : A7o 59.08%	Pockets : Q4o 49.41%	Pockets : 53s 39.07%
Pockets : A4s 58.36%	Pockets : J5s 49.36%	Pockets : T2o 38.69%
Pockets : QJo 58.34%	Pockets : 97s 49.16%	Pockets : 74o 38.47%
Pockets : A3s 58.21%	Pockets : T6s 48.58%	Pockets : 43s 38.21%
Pockets : K9o 58.17%	Pockets : K2o 48.36%	Pockets : 82s 37.92%
Pockets : K8s 58.02%	Pockets : 98o 48.08%	Pockets : 64o 37.65%
Pockets : JTs 57.99%	Pockets : T7o 48.08%	Pockets : 54o 37.49%
Pockets : QTo 57.69%	Pockets : 87s 47.82%	Pockets : 83o 37.12%
Pockets : K7s 57.62%	Pockets : Q3o 47.69%	Pockets : 73o 36.44%
Pockets : Q9s 57.59%	Pockets : J6o 47.63%	Pockets : 92o 36.12%
Pockets : A5o 57.44%	Pockets : J3s 47.5%	Pockets : 63o 35.9%
Pockets : A6o 57.23%	Pockets : T5s 47.29%	Pockets : 52s 35.69%
Pockets : K6s 56.41%	Pockets : J5o 47.22%	Pockets : 53o 35.25%
Pockets : A4o 56.31%	Pockets : 96s 47.05%	Pockets : 62s 34.74%
Pockets : K8o 56.24%	Pockets : Q2s 47.02%	Pockets : 72s 34.72%
Pockets : 44o 55.98%	Pockets : 86s 46.47%	Pockets : 43o 34.65%
Pockets : J9s 55.86%	Pockets : T4s 46.45%	Pockets : 82o 33.67%
Pockets : Q8s 55.75%	Pockets : T6o 46.32%	Pockets : 42s 33.65%
Pockets : Q9o 55.75%	Pockets : J4o 46.26%	Pockets : 32s 32.44%
Pockets : JTo 55.67%	Pockets : 97o 46.21%	Pockets : 72o 31.4%
Pockets : A2s 55.67%	Pockets : T3s 45.86%	Pockets : 62o 30.89%
	Pockets : 76s 45.81%	Pockets : 52o 30.81%
		Pockets : 42o 29.98%
		Pockets : 32o 28.98%

Security Testing

The application was tested using Visual Code Grepper distributed by the NCC Group. The Visual Code Grepper is an automated code security review tool that handles a multitude of languages, including Java. After testing Heads-Up Hold'em initially there was over 1000 Potentially dangerous code that was found during the testing, mainly associated with string modifiers.

After rectifying the potential risks found. The application's potentially dangerous code risk was reduced from 1395 to 615. The majority of the potential dangerous code left was located within the Grid class associated with the MigLayout. The consensus of these findings can be chalked down to false positives from the Visual Code Grepper.



The main classes utilized within the Heads-Up Hold'em application was found to have very low to none Potentially Unsafe lines of Code. The result of the security testing was very successful in ensuring the integrity of the application.

1.13 Evaluation

The final evaluation of the project can be summed up in various limitations that have been met throughout the application's development lifecycle.

Limitations of Software usage

Throughout the development of the software project, the references to the work of the "University of Alberta's Computer Poker Research Group" was paramount in reaching what was achieved, although the attempt in trying to adapt or simulate the classes they provided into my application was not the best course of action, as their artificial intelligent agent "Poki" was vastly more complex than what I could achieve within the limited timeframe.

Although time was wasted in trying to understand their class implementations, it was beneficial towards the continued development, once I decided to create my own classes. The end result being a program that works reasonably well, but which encounters a number of limitations.

Limitations of GUI

Initially I had high hopes for the creation of the applications GUI, attempting to mirror styles from other forms of poker application GUIs, such as Pokerstars, Paradise Poker and Full Tilt poker. But after understanding my restricted knowledge on the use of Java GUI widgets, such as Swing. I came to the realisation that it was more of an idealistic scenario rather than a realistic scenario.

After extensive research, and recommendations from various sources. I stumbled across MigLayout, which is a grid-based layout manager which uses Java Swing. Fortunately, with the use of MigLayout I was able to muster a very simplistic GUI, that works and although it doesn't essentially immerse the user, I am content with the outcome.

Limitations of AI Agent

Although the name of the Agent is "ProBot", which implies that it functions on high level of play, there are glaring issues in how the agent acts. Firstly, the agent will only bet the amount of the big blind when acting post-flop, ideally when betting I would have liked if I was able to implement a type of bet that varies in chip amounts. There is still a degree of randomness provided by the probability of how much the agent will raise by, once it has reached the river.

1.14 Definitions, Acronyms, and Abbreviations

Action. Refers to a player's action on the poker table. Actions may include folding, checking, calling, betting or raising.

All-in. When a player commits his entire stack of chips or money to the pot.

Big-Blind. In Heads-Up Texas Hold'em, the player who is not the dealer posts a forced bet called the big blind. This amount is set depending on the stakes being played. The amount of the big blind is usually twice that of the small blind, and equal to a small bet.

Small-Blind. In Heads-Up Texas Hold'em, the player who is holds the dealer button posts a forced bet called the small blind. This amount is set depending on the stakes being played. The amount of the small blind is usually half that of the big blind.

Bluff. Making a bet when holding a weak hand that has little chance to win if called.

Button. Also referred to as the 'Dealer marker'. The player with the button is the player just before the small-blind and has positional advantage over the entire table for the duration of the hand.

Community cards. Refers to the open cards dealt to the table in which players can all observe to make their hands.

Check-raise. A trap play in which a player checks in an attempt to encourage the opponent player to bet, and then raising at the next opportunity.

Flop. (a) The first three community cards dealt in Texas Hold'em. (b) The betting round that commences after the first three community cards have been dealt.

Turn. (a) The fourth community card dealt in Texas Hold'em. (b) The betting round after the fourth community card is dealt.

River. (a) The fifth community card dealt in Texas Hold'em. (b) The final betting round after the fifth community card is dealt.

Range. A collection of hands usually consisting of one or more hole-cards. This term is commonly used when an oppositions hands are not known, and can only be estimated.

Pot. The collection of all the betting amounts on the table at that point in time

Pre-flop. Refers to the betting round before the flop has been dealt.

Post-flop. Refers to the betting's rounds that extend during and after the flop has been dealt. This includes the flop, turn and river.

Hole Card. A player's private card in poker, unknown to the opponents.

used in balance with bets that are intended to be for value to create uncertainty about the

strength of their hand. If a player does

Aggressive. Refers to a player's style of play. An aggressive player will tend to raise and re-raise instead of calling. These players predominately raise or fold rather than flat calling.

Conservative. Refers to a player's style. A tight player plays comparatively fewer hands than a Loose player.

Loose. Refers to a player's style of play. A loose player plays comparatively more hands than the average player.

Limp. Refers to when a player chooses to simply flat call the pre-flop big blind amount.

Starting Hand. In Texas Hold'em, the first two-hole cards a player is dealt is known as the starting hand.

Straight. A poker hand ranked higher than three of a kind, but lower than a flush. Consists of five cards of differing suits in sequential order, e.g. 10, Jack, Queen, King, Ace

Straight Flush. The highest-ranking poker rank utilized by the ProBot agent, Consists of five cards of the same suit in sequential order, e.g. 10, Jack, Queen, King, Ace all hearts.

2 Conclusions

In conclusion, throughout the development of the project, there has been multiple positives and negatives encountered.

Even though the project involved understanding complex learnings and different aspects of Java I have never developed with. I managed to implement some form of agent that acts with some variations of logical decision making, which I initially intended prior to development.

Positive Observations

I thoroughly enjoyed researching the surrounding literature available on the topic of artificial intelligence poker domain is very extensive. The University of Alberta controls a strong foothold in leading the advancement of AI poker research, taking the time to observe the relevant academic throughout the project development was a great insight into the current state of Artificial Intelligence and Game Theory. A great amount of background theory from previous research was extracted and applied to the solution of this project.

Concepts such as hand evaluation algorithms were implemented into the no-limit variant of Texas Hold'em, where little research has been extensively done, so this gives me a great deal of satisfaction.

Although, the entire solution to the project was a large-scale operation, revolving around various complex areas of software, within the area of Artificial Intelligence. When reflecting on the outcome of the application I am genuinely happy with the result.

Most importantly, the project has improved my software development and programming skills. It has also improved my awareness of the importance of researching academic literature, continuous delivery such as including new features, configurations and bug fixes

Negative Observations

As previously stated above, the outcome of the Artificial Agent was satisfactory, although the main negative observation when reflecting on the outcome of the project falls on the effectiveness of some of agent's decisions are not as optimal as I would have liked.

The entire solution would have benefited if the agent acted with a fully optimised Hand Evaluator, unfortunately I was unable to implement an efficient Hand Evaluator for the agent, instead a HashMap was used for assigning values, which does not carry the same results as proficient hand evaluator would.

3 Further development or research

The application has multiple ways in which it could evolve over time. Some of which include, expanding to other variations of Texas Hold'em, such as six and nine table Texas Holdem, which would just see an influx of different AI agent types on the tables. Accomplishing this would definitely be harder to achieve, but I am confident that with enough time and research, I could implement these.

The application could also expand to different Poker games, such as Pot limit Omaha or Limit Poker, which are less popular games, but still carry a big following of players.

The agent's bet function does not work in the manner, that I would have liked. The agent will only bet a static amount that correlates to the value of the Big Blind. Unfortunately, I could not manage to get the agent to orchestrate a bet that differs at every hand. This will be a major improvement that I intend on changing for the future for the agent to act in a more sophisticated and interesting manner.

4 Bibliography

- AB, M. I. (2009). *MiG Layout Quick Start Guide* . Retrieved from MiG Layout Quick Start Guide : <http://www.miglayout.com/QuickStart.pdf>
- Baghirov, Z. (2018, May 13). *Creating a state of the art based poker player*. Retrieved from studentnet: <http://studentnet.cs.manchester.ac.uk/resources/library/3rd-year-projects/2016/zakir.baghirov.pdf>
- Cantero, P. (2016, October 09). *javapokertexasholdem*. Retrieved from Github: <https://github.com/phstc/javapokertexasholdem>
- Cappallo, T. (2017, November 30). *Development of a Heads-Up Autonomous Poker-Playing Agent*. Retrieved from Development of a Heads-Up Autonomous Poker-Playing Agent: http://by.tc/assets/cappallo_poker_project.pdf
- Colleges, H. a. (n.d.). *PokerRank*. Retrieved from Hobart and William Smith Education: <http://math.hws.edu/javanotes/source/chapter12/netgame/fivecarddraw/PokerRank.java>
- crazyjugglerdrumme, .. (n.d.). *#1 crazyjugglerdrumme*.
- crazyjugglerdrummer. (2009, July 26). *How To Make A Poker Game In Java*. Retrieved from dreamincode: <http://www.dreamincode.net/forums/topic/116864-how-to-make-a-poker-game-in-java/>
- Darse Billings, Denis Papp, Jonathan Schaeffer, Duane Szafron. (1998). Retrieved from Opponent Modeling in Poker: http://www.cs.virginia.edu/~evans/poker/wp-content/uploads/2011/02/opponent_modeling_in_poker_billings.pdf

- Darse Billings, L. P. (1999). *Using Probabilistic Knowledge and Simulation to Play Poker*. Retrieved from Using Probabilistic Knowledge and Simulation to Play Poker: <http://webdocs.cs.ualberta.ca/~darse/Papers/AAAI99.html>
- Davidson, A. (2002, July 24). *Package ca.ualberta.cs.poker*. Retrieved from Class Summary: <http://spaz.ca/poker/doc/ca/ualberta/cs/poker/package-summary.html>
- Davidson, A. (2002, July 24). *Poker Bot Artificial Intelligence Resources*. Retrieved from University of Alberta Computer Poker Research Group: <http://spaz.ca/poker/>
- Follek, R. I. (2003, May). *A Rule-Based System For Playing Poker*. Retrieved from CiteSeerX: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.459.2982&rep=rep1&type=pdf>
- Gabai, D. (2014). *PokerHand.java*. Retrieved from Princeton Education: <http://www.cs.princeton.edu/courses/archive/fall14/cos126/docs/PokerHand.java.html>
- Games, T. A. (2015). *Discussions: Texas Hold 'em*. Retrieved from The AI Games: <http://theaigames.com/discussions/texas-hold-em>
- Genuis, P. (2002, January 15). *Poker Genuis Forum*. Retrieved from Poker Genuis Forum: <http://poker-genius.com/forum/forum-10.html>
- Grom. (2009, February 03). *Building a Texas Hold'em playing AI..from scratch*. Retrieved from Stack Overflow: <https://stackoverflow.com/questions/506167/building-a-texas-holdem-playing-ai-from-scratch>
- Harvey, D. (2009, April). *An Environment to Develop a 6-handed No Limit Texas*. Retrieved from An Environment to Develop a 6-handed No Limit Texas: <http://www.cs.bath.ac.uk/~mdv/courses/CM30082/projects.bho/2008-9/Harvey-DB-Dissertation-2008-9.pdf>

- Immanuel Schweizer, K. P.-H. (2009, February). *An Exploitative Monte-Carlo*. Retrieved from An Exploitative Monte-Carlo: <https://www.ke.tu-darmstadt.de/publications/reports/tud-ke-2009-02.pdf>
- Kendall, G. (2002, May). *Texas Hold 'Em*. Retrieved from "Texas Hold 'Em: <http://www.cs.nott.ac.uk/~pszgk/ugyear2/projects/2001-02/dissertations/gp-gxk2.pdf>
- Kullmann, O. (2009, November 05). *java Poker N*. Retrieved from Swansea University : http://www.cs.swan.ac.uk/~csoliver/ProgrammingJava201011/CS-M41_Programs/Courseworks/200910/One/Poker.java
- Kumar. (2012). *Creating a NL Texas Hold'em Bot*. Retrieved from Creating a NL Texas Hold'em Bot: <http://cs229.stanford.edu/proj2012/Kumar-CreatingTexasHold%27EmBot.pdf>
- Mccurley, P. (2009, May 08). *An Artificial Intelligence Agent For Texas Hold'em Poker*. Retrieved from Poker-AI: <http://poker-ai.org/archive/pokeraai.org/public/aith.pdf>
- McLeod, J. (2013, July 15). *Ranking of Poker Hands*. Retrieved from Pagat: <https://www.pagat.com/poker/rules/ranking.html>
- McLeod, J. (2013, July 15). *Rules of Poker*. Retrieved from Pagat: <https://www.pagat.com/poker/rules/#Basics>
- Mendes, P. D. (2008, July). *High-Level Language to build Poker Agents* . Retrieved from High-Level Language to build Poker Agents : <https://pdfs.semanticscholar.org/f165/b5608f0f0a71206ff9dd83c6ccb6167966fb.pdf>
- Michael Bowling, N. B. (n.d.). *Heads-up Limit Hold'em Poker is Solved*. Retrieved from Heads-up Limit Hold'em Poker is Solved: <https://webdocs.cs.ualberta.ca/~bowling/papers/15science.pdf>

- Nick Abou Risk, D. S. (2010, May). *Using Counterfactual Regret Minimization to Create*. Retrieved from Computer Poker Research Group:
<http://poker.cs.ualberta.ca/publications/AAMAS10.pdf>
- Papp, D. R. (1998). *Dealing with Imperfect Information in Poker*. Retrieved from University of Alberta: <http://poker.cs.ualberta.ca/publications/papp.msc.pdf>
- PokerHand*. (n.d.). Retrieved from University of Georgia:
<http://cobweb.cs.uga.edu/~gtb/1302/Project1/PokerHand.java>
- Robert Sedgewick, K. W. (2018). *Hash Tables*. Retrieved from Princeton Education:
<https://www.cs.princeton.edu/courses/archive/spring18/cos226/lectures/34/HashTables.pdf>
- Schuijtvlot, E. (2011, June). *Application of AI in poker*. Retrieved from BMI paper:
https://beta.vu.nl/nl/Images/werkstuk-schuijtvlot_tcm235-225501.pdf
- Science, C. C. (2002, July 30). *PokerHand*. Retrieved from Cornell Computer Science:
<http://www.cs.cornell.edu/courses/cs100/2003su/assignment5/solution/PokerHand.java>
- Stigter, O. (2016, August 12). *texasholdem-java*. Retrieved from Github:
<https://github.com/ostigter/texasholdem-java>
- Stigter, O. (2016, August 12). *texasholdem-java*. Retrieved from Google Code Archive: <https://code.google.com/archive/p/texasholdem-java/>
- The Computer Poker Research Group, U. o. (2017). *Computer Poker Research Group*. Retrieved from Computer Poker Research Group:
<https://webdocs.cs.ualberta.ca/~games/poker/>
- West, M. (2007, January 04). *Programming Poker AI*. Retrieved from Cowboy Programming: <http://cowboyprogramming.com/2007/01/04/programming-poker-ai/>

Wiki, L. (2015, May 07). *LWJGL library*. Retrieved from LWJGL library.:
<http://wiki.lwjgl.org/index.html>

5 Appendix

5.1 *Project Proposal*

The main objective is to develop an Artificial Intelligence agent that is capable of good decision making when playing Texas Hold'em poker.

In doing this, the objectives are comprised of 5 points.

- To investigate the characteristics strong poker players, possess and compare these results with the agent solution.
- To measure the performance of the agent against human opposition over a multitude of hands, and document these results.
- Design an efficient Hand Evaluation program
- Design the agent to play no-limit poker variant of Texas Hold'em
- To produce a finalized agent, that produces positive results over a certain number of hands.

The game of poker offers a well-defined domain in which to investigate some fundamental issues in Artificial Intelligence, such as how to handle a game tree that presents *Imperfect Knowledge*, through the means of concealed opponent's cards.

This project will aim to investigate what Artificial Intelligence techniques can be applied to the domain in order to play up to a human standard of decision making. This will hopefully result in creating a bot that plays Texas Hold'em in a profitable manner.

To simplify the problem, some limitations to the game are made. Firstly, though this is No-Limit Hold'em, the bot's bets will be fixed. A common raise made by many players is 3 times the bet made in front of them. If the bot is leading out, the bet will be 2/3 of the pot. Lastly, there will be no concept of bankroll, however I will aim to track the winnings and losses of the bot.

5.2 Background

In the domain of Artificial Intelligence there has been a plethora of games that have been solved to date. Some examples of these agents are, IBM's "Deep Blue" for chess, The University of Alberta's "Chinook" for checkers and Michael Buro's "Logistello" for Othello.

These agents have effectively solved those games and have beaten the best human minds in the world, demonstrating the power of computational processing. However, what all these games have in common is, they are all "perfect information" games. Perfect information games refer to the game in which each player, at any point in the game has complete knowledge of the current game state. Games like Chess, Checkers and Backgammon are "perfect information" games. Players who play these have perfect knowledge of the game state as they can see all remaining pieces on the game board.

Well known game-search trees, such as alpha-beta search can be used to explore deep into the game tree to find and choose the worse-case action the opponent cannot compete against.

In contrast to this, Poker is a "imperfect information" game, this means that certain information within the game is private, in terms of poker, each player receives private cards. As a result, no player can know the current position in the game tree.

Poker is a non-deterministic game. A player's actions within the poker domain can never guarantee the same outcome.

Poker has stochastic outcomes. This element of change through random shuffling of the cards creates uncertainty and adds a great deal of variance to the results.

Texas Hold'em is a poker variation that uses community cards. This variant of Poker was chosen because its rules have specific characteristics that

allow new developed methodologies to be adapted to other Poker variations with reduced effort.

Rules

At the beginning of every game, two cards are dealt to each player. The dealer player is assigned and marked with a dealer button. The dealer position rotates clockwise from game to game. After that, the two players to the left of dealer post the blind bets. The first player is called small blind, and the other one is called big blind. They respectively post half of minimum and the minimum bet. The player that starts the game is the one on the left of the big blind. One example of an initial table configuration is shown in Figure 2. The dealer is the player at seat F and the small and big blind players are respectively the A and B seats.

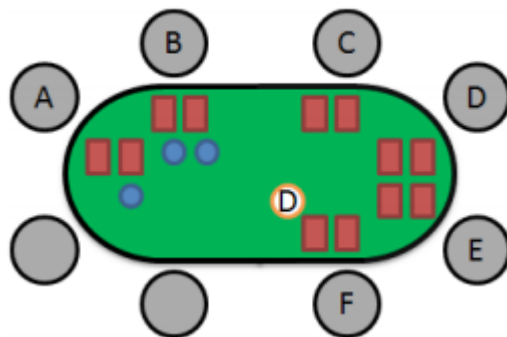


Table Layout

The first player to act is the player to the left of the big blind (Player C) And the next player is the closest one to the left of the current player. Each player can choose one of the following actions

- Call: Match the current highest bet
- Raise: Bet higher than the current highest bet
- Fold: Forfeit the hand, thus give up the pot

There are four betting rounds in Texas Hold'em, where each round new community cards are revealed.

- Pre-Flop: No community cards
- Flop: three community cards revealed
- Turn: The fourth community card is revealed
- River: The fifth community card is revealed

After the river, if at least 2 players agree to call the pot, the showdown round comes. This is when all players may show their cards and the one with the best hand wins the pot. If players have similar ranked hands, there is a tie and the pot is divided. This is otherwise known as a “chop-pot”

Hand Rankings

A poker hand is a set of five cards that identifies the score of a player in poker. The hand is made by combining the player's personal cards with the community cards. The table below presents the ranking of each combination with a short description.

Hand Description	Hand Example
Royal Flush: this is the best possible hand in standard five-card Poker. Ace, King, Queen, Jack and 10, all of the same suit.	
Straight Flush: Any five-card sequence in the same suit.	
Four of a Kind: Any set with four cards with the same rank.	
Full House: Three cards with the same rank plus two cards with the same rank.	
Flush: Any set with five cards of the same suit, but not in sequence.	
Straight: Five cards in sequence, but with different suit.	
Three of a kind: three cards with the same rank.	
Two Pair: Two separate pairs, and one kicker of different value. The kicker is used to decide upon a tie of the same two pairs.	
One Pair: Two cards with the same rank and three kicker cards.	
High Card: Any hand that does not qualify as one of the better hands above. Ranked by top card, then the second card and so on.	

Hand Evaluation Algorithms

The algorithm in which that is used to quantify the agent's Hand Strength, regardless of all cards being dealt. This algorithm is key, as it considers all the possible better hands the agent could have, the same, and all the worse hands at the point of calculation. The algorithm iterates through all possible starting hands and returns a percentage as a result.

```
Function HandRank(Hand) {  
    Sort(Hand);  
    If IsStraightFlush(Hand) Return 9;  
    If IsIsFourOfAKind(Hand) Return 8;  
    If IsFullHouse(Hand) Return 7;  
    If IsFlush(Hand) Return 6;  
    If IsStraight(Hand) Return 5;  
    If IsThreeOfAKind(Hand) Return 4;  
    If IsTwoPairs(Hand) Return 3;  
    If IsOnePair(Hand) Return 2;  
    Return 1;  
}
```

Hand Potential

Hand Potential is an algorithm that calculates the possible evolution of the hand quality throughout the game. In Texas Hold'em, when the game reaches the Flop round, there are still two more community cards to be revealed. This means that the current hand rank may improve, since the hand is composed of the set of five available cards that has the highest rank among all available cards. This is an extension of the hand evaluation, but instead of only considering the current available cards, it considers the possible community cards that have not been revealed yet. This also considers that the opponent's hands might improve as well.

Non-Deterministic Game

Non-deterministic games are often described as games with an element of chance. These games do not result in predictable outcome. Examples of such games are Backgammon and Poker (their source of chance is dice and card respectively). What makes these games different from deterministic games are the additional nodes called 'Chance' or 'Nature' in their game trees.

Imperfect Information

Imperfect information game corresponds to the game in which certain information is private, meaning that other players cannot see it. For example, in Poker each player receives private cards. As a result of this, no player can clearly know the current position in the game tree.

The utility or payoff

The utility in the game is the expected value when a round of a game is played. In the poker game, it is the number of chips that was acquired or lost at the end of the hand (round).

Nash Equilibrium

Nash equilibrium is a strategy profile σ where no player can increase their utility by unilaterally changing their strategy (Johanson, 2007): This means that for player 1, there is no other strategy in Σ_1 that would produce more utility against σ_2 than its strategy in σ . The same is true of player 2. (Johanson, 2007)

5.3 Technical Details

Implementation

This application will be developed using an Agile software development process. The specific agile practice I intend to adopt for this project is Iterative Development. The main idea behind Iterative Development is to break down the whole project into smaller parts or iterations. This was chosen with the goal of completing significant parts of the project at the end of each iteration. I believe this will be proving to be beneficial as it allows for the development of the more

Java

Java is the programming language that will be used to develop the application. This is because Java allows applications to be easily integrated with web application using Java Applet, another reason behind using Java was for future extension of the application to be ported to mobile, using Android application development which uses Java. The object orientated nature of Java, helps in separating the features of the application for easier management and debugging.

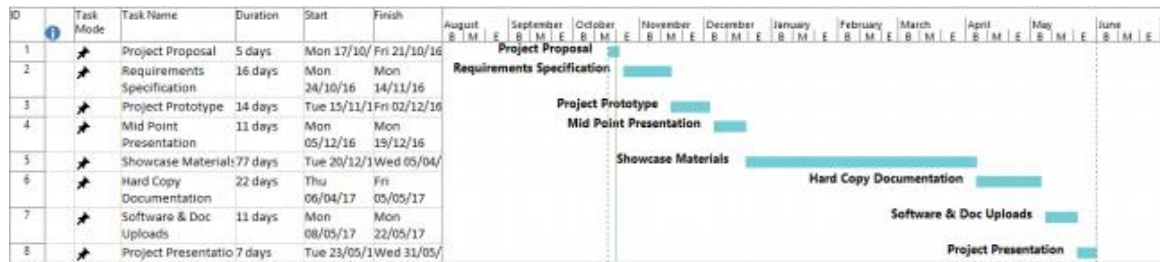
MigLayout – Java Layout Manager

MigLayout is a grid-based layout that allows for designing complex GUI layouts for Swing applications with ease.

MySQL

MySQL was the primary relational database management system taught to this year's 4th Year NCI computing students during their time at college. Thus, it made sense to use MySQL for this project rather than having to learn an entire new database system.

5.4 Project Plan



5.5 Monthly Journals

September Journal

Student Name: Lee Murray

Programme: BSc Computing

Month: September 2017

When attempting to come up with an idea for my final year project, I wanted to do something aligning to my cyber security stream, although I was unable to come up with a solid idea in time for the pitch so instead of going in and saying I don't have an idea, I pitched a Poker application which included an Artificial Intelligent agent, which I presumed would get rejected. To my surprise it got accepted. I have a keen interest in Poker and have played it for a couple of years, although developing an AI agent seems as an ambition task, I'm positive it will also be very interesting.

Supervisor Meetings

I didn't have a supervisor meeting during the month of September

October Journal

Student Name: Lee Murray

Programme: BSc Computing

Month: October 2017

I am very glad to say that, my project idea was approved by the college committee, who queried me about the application that I want to build. So, this month my achievements were: First, I prepared for the presentation Pitch and then I presented my project idea in front of the college committee. Second, I started and completed the project proposal and then I have uploaded it to Moodle on time. Thirdly I began working on my Requirements specifications. I also did some research on the topic of Game Theory in Artificial Intelligence and Bots in Poker

Supervisor Meetings

I didn't have a supervisor meeting during the month of October

November Journal

Student Name: Lee Murray

Programme: BSc Computing

Month: November 2017

During October, I met with my Supervisor Vikas, whom pointed in a direction to find the best programming language I can use to develop the algorithm for the AI agent, with the use of document surveys. As I already have previous knowledge of Java from using it in college, I decided to stick with what I know, rather than take on a new language in this already difficult enough task. We also discussed about the platform of the project, whether to develop it as a NetBeans application or on Android. Vikas recommended doing it as a NetBeans application to keep it as simple as possible as I have not worked with Android studio before. Although he offered a good suggestion to include on the extensibility of the application that as further development takes place, using Android studio is a possibility.

My Achievements

Finished the Requirements Specification

Supervisor Meetings

Date of Meeting: 14th November Items discussed: Discussing Project Idea, what language and platform to use.

December Journal

Student Name: Lee Murray

Programme: BSc Computing

Month: December 2017

December was an extremely busy month for all modules, unfortunately my software project needed to be pushed back on my priority list, in order to finish off other projects for other modules.

My main focus in regard to the software project is on the mid-point presentation. Although I was happy with the slides I provided and how I articulated them, my prototype did concern me as I could only manage to provide a small amount of functionality which comprised of a deck shuffling method and a mock-up GUI.

I did hope to have some form of GUI developed in Java Swing by this time, but the decision to simply have a mock up to show what I'm trying to do would be best I could achieve at this moment in time.

I believe the presentation went well, there were little questions asked which I took as a positive to my presentation showing in full the current and future development which I aim to achieve.

Supervisor Meetings

I had a meeting with Vikas prior to my mid-point presentation, he offered me excellent advice and what I should be focusing on to gain the maximum marks.

He also instructed me to explicitly focus on my exams after the presentation, which gave me a sense of relief and reassurance, as I was unsure if I should be still giving excess time to my software project during my exam time to not fall behind.

Items Discussed:

- Project Prototype
- GUI
- Deck Shuffling method
- Slide preparation

My Achievements

The midpoint presentation went very well from my perspective. I received helpful feedback from the supervisors regarding the development of the Poker agent which I gladly took on board.

January Journal

Student Name: Lee Murray

Programme: BSc Computing

Month: January 2018

After the completion of my exams, I made a decent head start on the development of the GUI for the user's configurations of the application. I came across a grid-based layout called MigLayout library which utilises Java Swing. MigLayout offers a vast number of tutorials and examples, which aided me in developing the UI of the application.

Solid progress was made on the core classes of the application, such as the card, player, deck and suit classes.

Supervisor Meetings

No meetings scheduled during the month of January.

February Journal

Student Name: Lee Murray

Programme: BSc Computing

Month: February 2018

February brought with it a great deal of progress on the development of the Artificial Intelligent agent. From the previous GUI work in January, I was able to set up an environment for the agent to act in, this allowed me to further enhance the algorithm and methods it follows, and thus improve its actions and decision making.

The initial development of the applications controllers for a game such as Texas Hold'em were started this month, these included abstract classes that contained all relevant information that are required to simulate a game of Texas Hold'em, such as the players involved, the big and small blind amounts and how the betting occurs in a sequential order.

Supervisor Meetings

Date of Meeting: 22nd Feb

Items discussed:

- Hand Evaluator implementation
- GUI Improvements
- Required Agent methods

March Journal

Student Name: Lee Murray

Programme: BSc Computing

Month: March 2018

During the month of March, I took a break from working on Agent methods and focused more on UI development and User registration and login forms, along with the security related features.

With continued research of experimenting with using the MigLayout grid, I developed a test UI for user configuration, including the Player Menu, Betting menu and an Appearance menu.

The player menu is aimed to allow users to choose from different types of agents I intend on allowing the player to play against and choose how many chips they wish to play with, this will ultimately decide on the pace of the game they would like to play.

The betting menu will give the user the ability to change the number of raises allowed by each player during every hand.

The user should also be able to change the amount of chips the big and small blind are, as well as implement antes during the game.

Creating secure registration and login form for the user is very important for the integrity of the application as well as being a cyber security student, I felt it was important for the application to have some element of security related feature contained within it.

The associated security element with the registration form made use of a MD5 hash of the user's password that is stored within the database.

I also included a log file that will handle errors that occur in the registration and login form.

Supervisor Meetings

Date of Meeting: 12th March

Items Discussed:

- Final Preparation
- Project progress on basic functionality