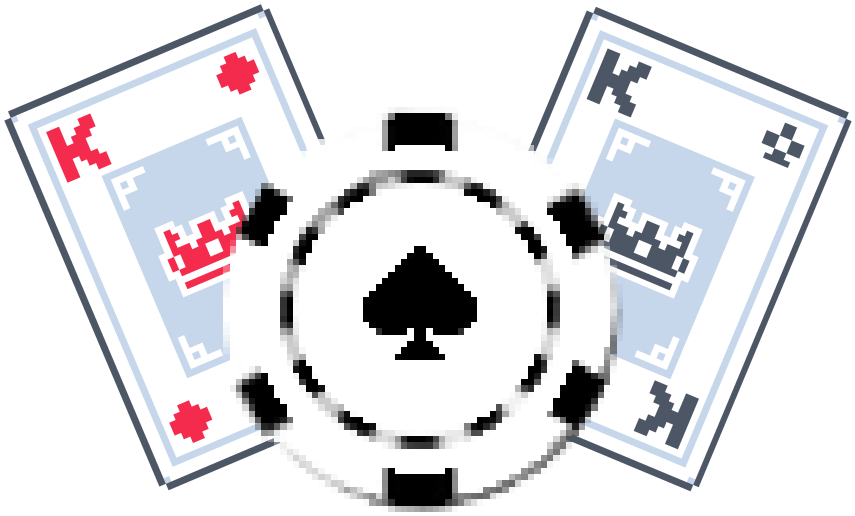
PIXEL POKER



Centre Number: [redacted]

Centre Name: [redacted]

Candidate Number: [redacted]

Candidate Name: [redacted]

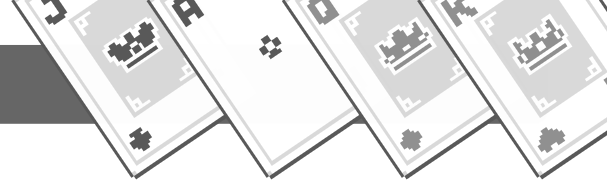
Contents

Analysis 6
Introduction 6
User Interaction 7
General Questions 7
Non-Poker Players 9
Poker Players 10
Existing Solutions 12
Red Dead Redemption 2: Poker Mini game 12
Prominence Poker 14
Objectives 26
Objective Reasoning 31
Main menu: 31
Single player: 31
Multiplayer: 31
In-game character: 31
Bots: 32
Poker assistance: 32
Game settings: 32
Art style & sound design: 32
Communication: 32
Save Data: 32
Game System 33
Time Constraints 33
Design 34
Prototype 34
Reasoning 34
Overall 34
Prototype Code 37
Menus 42
Reason and need for menus 42
Menu Design 42
Menu Flowchart 46
UI Elements 47
Button 47
Button Flowchart 47

Image 48
Image Flowchart 48
Text Input 48
Text Input Flowchart 49
Text Label 49
Text Label Flowchart 50
Main 50
Purpose and Function 50
Initial Class Diagram 50
Compare Module 51
Purpose and Function 51
Flowchart 53
Asset loader Module 54
Purpose and function 54
Client Module (networking) 55
Client Networking 55
Client Networking Flowchart 56
Server Module (networking) 56
Server Networking 56
Server Class Diagram 57
Server Networking Flowchart 58
Bots 59
Technical Solution 60
Key Algorithms and Classes – main.py 60
Overview 60
Main Class 60
Game Class 65
Card Class 66
Button Class 68
Text Input Class 69
Image Class 70
Text Label Class 71
Player Info Class 71
Main Player UI Class 73
Key Algorithms and Classes – compare.py 81
Overview 81

sortRule() 81
concatenateInts() 81
getValueOfHand() 81
valueToName() 84
Key Algorithms and Classes – assetloader.py 85
Overview 85
imageGroups 85
load() 87
Key Algorithms and Classes – client.py (networking) 87
Overview 87
packetID 87
Client Class 88
Key Algorithms and Classes – server.py (networking) 92
Overview 92
packetID 93
Server Class 93
Server Bot Class 103
Server Client Class 109
Deck Class 110
Card Class 111
Objectives 113
Testing 122
Menu Testing 123
Singleplayer Testing 125
Multiplayer Testing 128
UI Testing 131
Testing Fixes 133
Test 2.11 133
Test 2.14 133
Test 3.11 134
Test 3.14 135
Evaluation 137
Objectives 137
Feedback 146
Client Feedback 146
My thoughts and improvements 146

Improvements that could have been made if there was more time. 146
Appendix 148
Removed Data 148
Question 4 – Removed Responses 148
Question 7 (Poker Players) – Removed Responses 148
Complete program code 149
main.py 149
compare.py 169
assetloader.py 172
client.py 174
server.py 180



Analysis

Introduction

Within the past year or two, I learnt to play Poker specifically one on the most played versions of Poker called “Texas Hold ’em”. So, I know the basic rules and strategy of Poker. Additionally, in the past year I have also learnt how to use the Python module named “Pygame” which allows you to make games with relative ease. I wish to combine both these hobbies which I have into a game, using Python and Pygame. I could use other programming languages, which might be better in some cases, but as I know Python the best, I will use it. One of my main reasons for choosing Python as my language of choice is I can easily get help if I need to, etc.

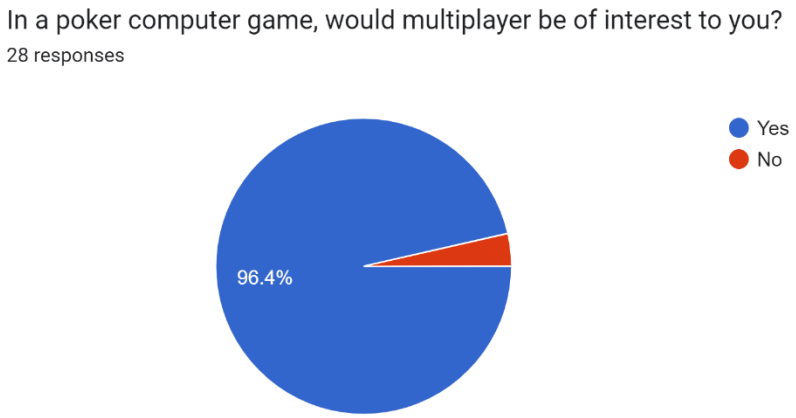
My aim is to create a 2D Poker game in Python using Pygame. I am hoping to take some inspiration from existing poker games like the poker mini games from Red Dead Redemption 2 or Rust, and from games where it is solely about poker like Prominence Poker or Poker Night or even open source/indie developed poker games. With these games I can try to see how they solved specific problems and what features they do or don’t have.

User Interaction

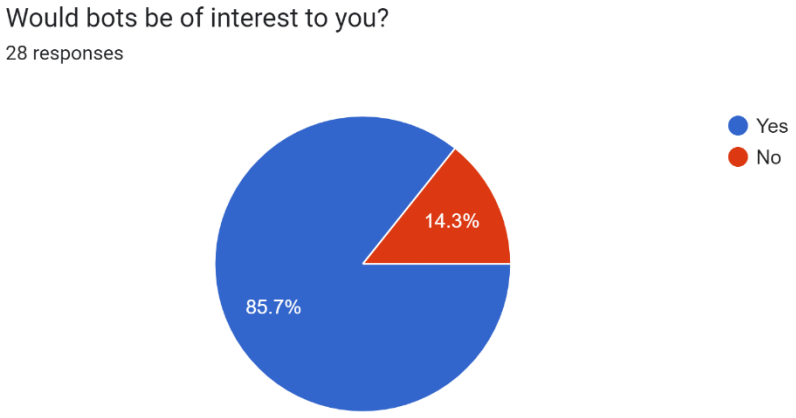
In order to get a better understanding of what my users would like out of a poker game, I asked them some questions. I used Google Forms to write up my questions to get results. In my form I asked questions which would be valuable for my research, for both people who play poker and those that don't.

General Questions

To start with, we have the general questions. Which both people who play poker and those that don't answered. These questions aren't specific to poker, but just a game in general, hence the fact that both parties answered.



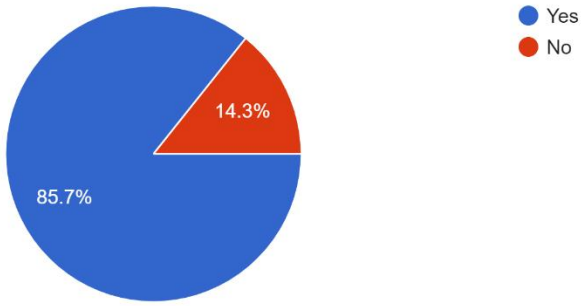
Clearly with such a high majority, multiplayer needs to be in my poker game. With multiplayer being the highest, this shows me that in order to target my users' needs I would need to add multiplayer to my poker game.



This data shows to me that in order to target my selected audience, I would need to add bots in to my game. This is because 85.7% of my surveyed users believed that bots would be beneficial to my game.

Would a chat system be of interest?

28 responses



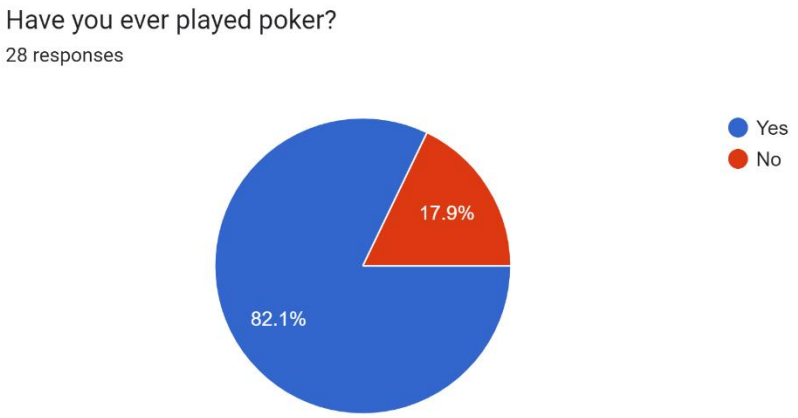
This data is almost identical to the previous question, the only differences being that the question is different, and that people from the previous question might have not selected the same option. If this is anything to go by, then a chat system should be joint priority with bots.

What else would you expect from a poker game?

VOIP so you can feel like you are really at a poker table Also give the bots a difficulty
Perhaps a ranking system
Public/Private servers, customizable themes
Ease of use
make borrowing money from other players a thing, but it comes with some tax, like u pay back 110% at the end of the round, if u can afford it, and also its a choice for the other player to accept if they will give
Win percentages for hands
Tally of wins
in game currency and stuff to do with your winnings
Maybe a point based that indicates someone's rank for example the more games they have one the higher their points or rank.
a game of betting and desiving your friends
chips, rebuys, timer, in game currency, texas hold'em mode(different gamemodes/versions)
unlockable items

(some responses have been removed from this table and have been added to the appendix under 'removed data')

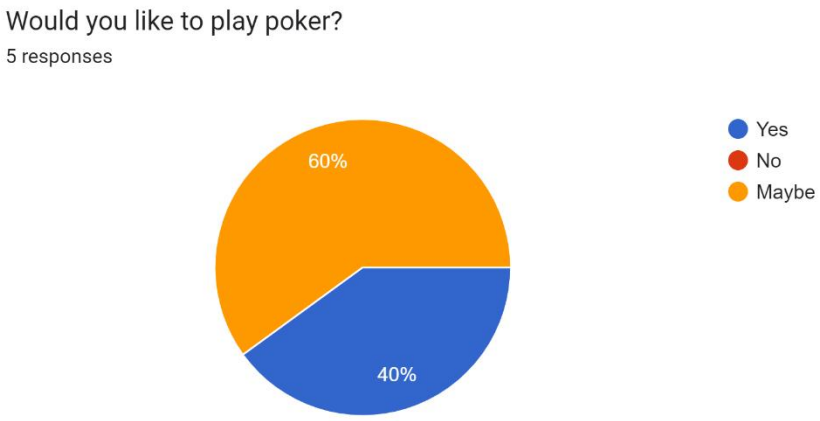
These responses allowed me to double check if what I wanted to add is of by demand and make sure that I don't miss obvious features. Like *'ease of use'* for example. However, some of the response helped to give me ideas for features I would have never thought about, like a VOIP system or displaying win percentages for certain hands. So overall, I got some positive, well needed results for my poker game.



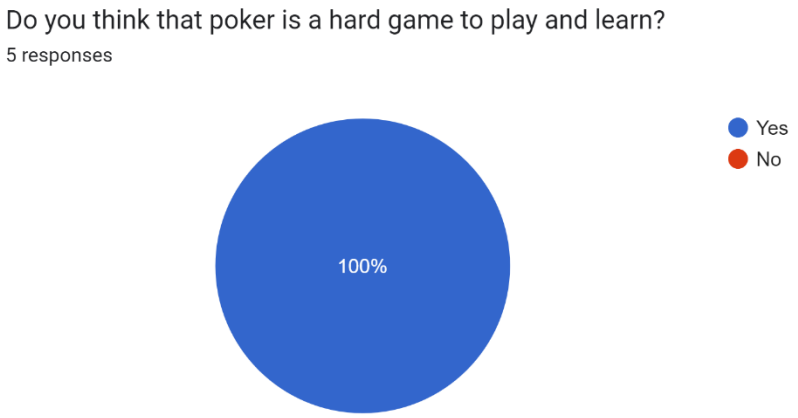
This data allow me to see what percentage of people I surveyed have actually played poker with allows me to see roughly how many people have played it before.

Non-Poker Players

These questions are only given to people who responded with 'No' to the previous question. Therefore, these results are aimed at people who have never played poker, asking question relating to what they think about poker and what they think could make a poker game better.



This data is valuable to me as it shows me if trying to target/cater my poker game to people who have never played the game is worth it or not.



The data above shows me that if I were to make the decision of also targeting non-poker players, I would need to make it easy and clear to understand the rules of poker like allowing users to see a chart of what hands there are and their ranking.

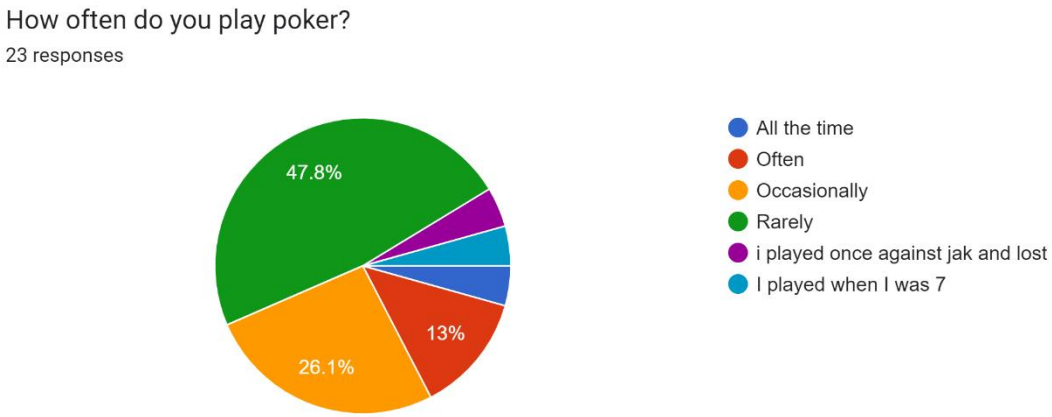
What do you would make poker easier to explain to someone?

Step by step tutorial
An instruction sheet
A tutorial that explains different features in the game and has some test games against bots
A cheat sheet with all the card combinations you want to get. A learning mode where you can see everyone's cards to help you understand how the game works

These responses help me to give some ideas on what features I can add to the poker game to make it more understandable for newer players.

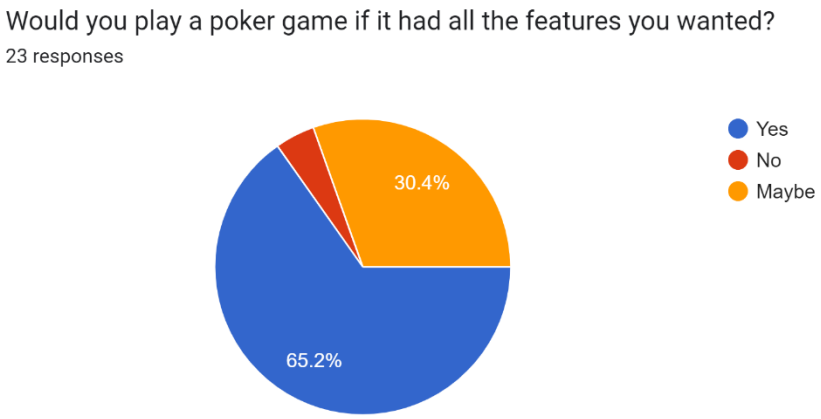
Poker Players

This section is only for people who responded to question 5 with a 'Yes'. These question are aimed to get a better understanding of what poker players want from a poker game. Relating to their experiences in poker.



This shows me how often people who play poker actually play for. The data shows that most people rarely play poker with only about 13 percent of players play often. This could

mean that the majority of these players don't find poker interesting to play or simply do not have the time to play. This gives me something to focus towards – making the game more open to people and allow them to play as often as they wish.



This shows me that if I do create a poker game with features which people request then a majority of player would play my poker game with only a small handful of people saying they wouldn't play.

Apart from Texas Hold 'em, have you ever played different version of poker that you would like to possibly see in a game?

no, im too bad at poker to really say i'd want to see loads of different ones
I haven't played a different version.
speed rounds(the timer has less time on the clock)
blackjack even though technically its not poker

The nature and lack of responses compared to the rest tells me that other gamemodes are not necessary as many players do not know other modes of poker such as three-card poker etc and are only accustomed to typical Texas Hold-em poker. Therefore, adding other game modes is unnecessary.

Existing Solutions

There are many different games that either revolve around poker or have poker mini games within them. These games focus on many different things, like how in some poker-based games there is a global money, levels, cosmetics. Whereas other games that have poker as a mini game usually have very basic features. Looking at other games that involve poker or have poker in them helps me to gain an understanding of what solutions and techniques other developers have overcome and how they have done so. Looking at these other examples helps me to gain an understanding of features which are typical of poker and, and some which are more bespoke and unique to one. Using the information I gather from this, I can use it to aid in my program's development.

Red Dead Redemption 2: Poker Mini game

Red Dead Redemption 2's version of poker is very bare bones and simple, no extra gimmicks like text or voice chat, just simple poker. In this version of poker, you play with other people in the online version whereas in single player you play against AI.

This image shows us the main look and feel of the game, a classic green felt covered table with players dotted around with their cards and chips displayed on the table. On the left-hand side, the game displays the player information like their position in the game, dealer, little-blind. The game also displays a picture of the player along with their current winnings and the last action they did, call, check, raise, etc.

On the right-hand side, the game displays the controls and the corresponding key for that action, like calling, raising, or checking plus a few other controls for other actions. The game also shows the currently drawn community cards in the top right-hand corner of the screen along with the total pot.

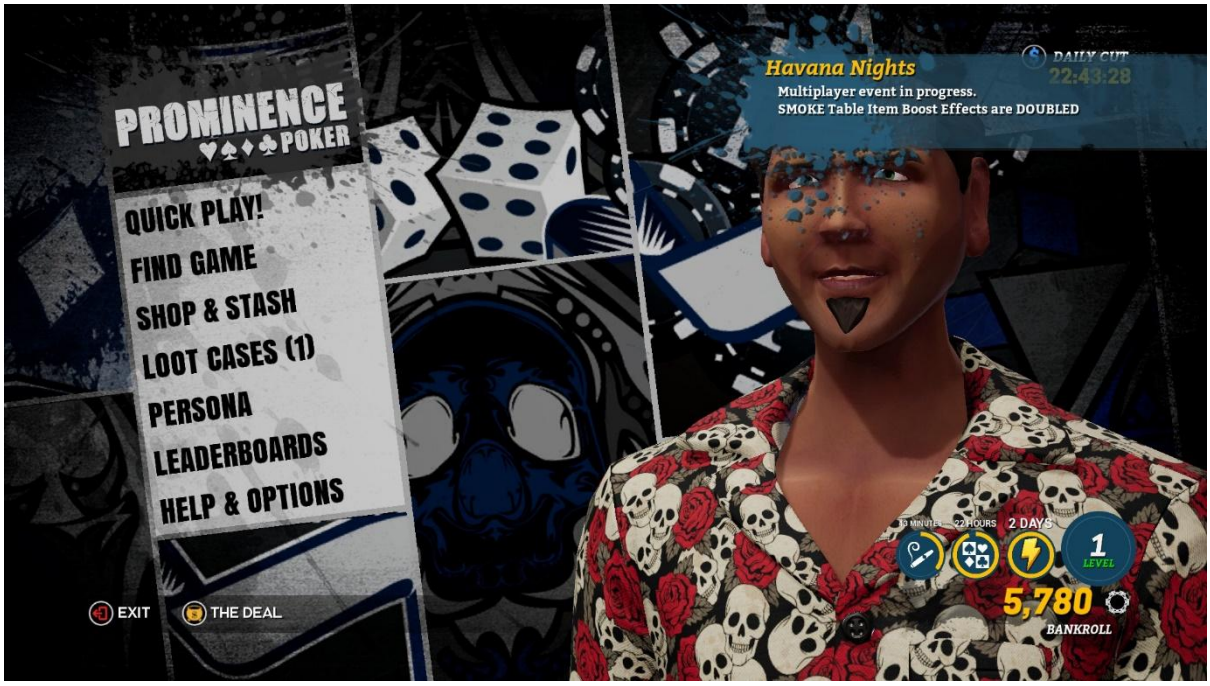
When you look at your cards, the game places you in first person so that you can look at the card in your hand as the game is played from the third-person perspective. The game also displays what hand you have in the top left, for those players who are unexperienced or don't want to spend time figuring out what hand they have. Actively having to look at your cards making you feel more actively involved in the game, making it feel more 'real'.

Here, you can see the community cards being drawn out, they are being physically drawn out on the table. When this happens the game forces all the players in the game's camera perspective to focus on the community cards as they are drawn out. Whilst this is happening, the community cards in the top right are updated with simple easy to understand icons of the cards.

When pressing the 'H' key in RDR2 on the PC version, in poker it brings up a Graphical User Interface (GUI). This GUI shows the player the different poker hands that are possible, and what cards you need. The only real negative I have of this is that, if you are new to poker, you would think that to get a pair for example, you would need a 5 of diamonds and a 5 of clubs. However, this GUI makes the game more accessible to people who either rarely play or don't play at all.

Prominence Poker

Prominence Poker is a game based entirely around poker. However, it does add a lot of features – that being mainly social – rather than alter the game of poker in anyway. The poker in the game works via Texas hold-em rules and is completely unchanged. The main features this game brings to the table when compared to other poker games is the social features such as character customisation and interactions.



This is the main menu for the game, the menu is clear to read and designed with poker-related graphics surrounding it. Due to the fact that this is a game entirely based around poker, the game has global money. This means winning money in one game would benefit you overall as it would increase your overall money. This information is displayed to the user in the bottom right as 'Bankroll'. Every day, users may receive a 'daily cut' which is a sum of money which is added to the players 'Bankroll'. This stops users from becoming bankrupt and therefore being unable to play.

The two main buttons users would be pressing is 'Quick Play' where users are automatically joined to an ongoing game, or find game where users may create their own lobby, join one which is about to start, or play a single player game against bots. However, despite their being single player it is mainly there as a place where newer players can learn about the game and poker in general in a quiet/controlled environment as the game is based on player between player interactions.

The 'shop and stash' button allows users to open up a shop on their screen displaying various different cosmetics etc that the user is able to purchase with their overall chips. This includes clothing for your in-game character and items in the game such as 'Table Items' which are items placed on table that other players can see and can see you interact with. The shop allows users with high amounts of chips to spend their winnings therefore adding some level of progression to the game helping to increase overall player retention.

The 'Persona' button is a button which sends users to another menu where they may look at different parts of their overall in-game character, this include character customisation and statistics about your character.

The 'Leader boards' button allows the user to look at leader boards showcasing the players with the most amount of money in the entire game. This can give users another incentive to play rather than just to level up their character. Some users will want to be at the top of the leader board and compete for number one, thus driving up the player retention.

Finally, the 'Help and Options' menu is where users are able to change settings of the game such as graphics and other settings such as resolution etc. The player, if they are unsure about a certain part of the game, they may be able to find an answer to it in the help section. For example, if they were unsure about the rules of poker.

The player also has the option to quit the game from the main menu if they feel the need to do so.

The players in-game character is displayed on the right-hand side of most menus in the main menu. This helps users know exactly what their character looks like and therefore help the user make informed decision when it comes to character customisation and appearance.

Additionally, if the user has never played 'Prominence Poker' then upon entering the main menu they would be asked if they would like to play the tutorial level. This is good as it helps newer players who either don't know poker very well or even experienced players a chance to learn the game in a controlled environment against bots.

Here the user is able to select three options relating to 'Persona'. These being 'Rap Sheet', 'Character', and 'Affiliation'. The rap sheet button just opens menu where they are able to see certain statistics about their character in game. The character button opens a menu where players can customise their character. This includes clothing and appearance as well as a in game nickname. The 'Affiliation' button opens a menu where players are able to see the in-game level and information relating to that level.

Here the player is given an overview of their character. Their in-game player model is displayed in full on the left, giving the user a better idea what they look like in game compared to other players. This menu also displays 'Challenges' which the user can do the gain extra in-game experience and rewards. The players overall money is also displayed here as well as their level and the recent unlocks because of their level. On the top, their current ratings are displayed as well giving the player a better idea of what kind of poker player they are.

Here the player is able to change the appearance of their entire character. Such as clothing, nickname, gender, head, and hair as well as overall body type.

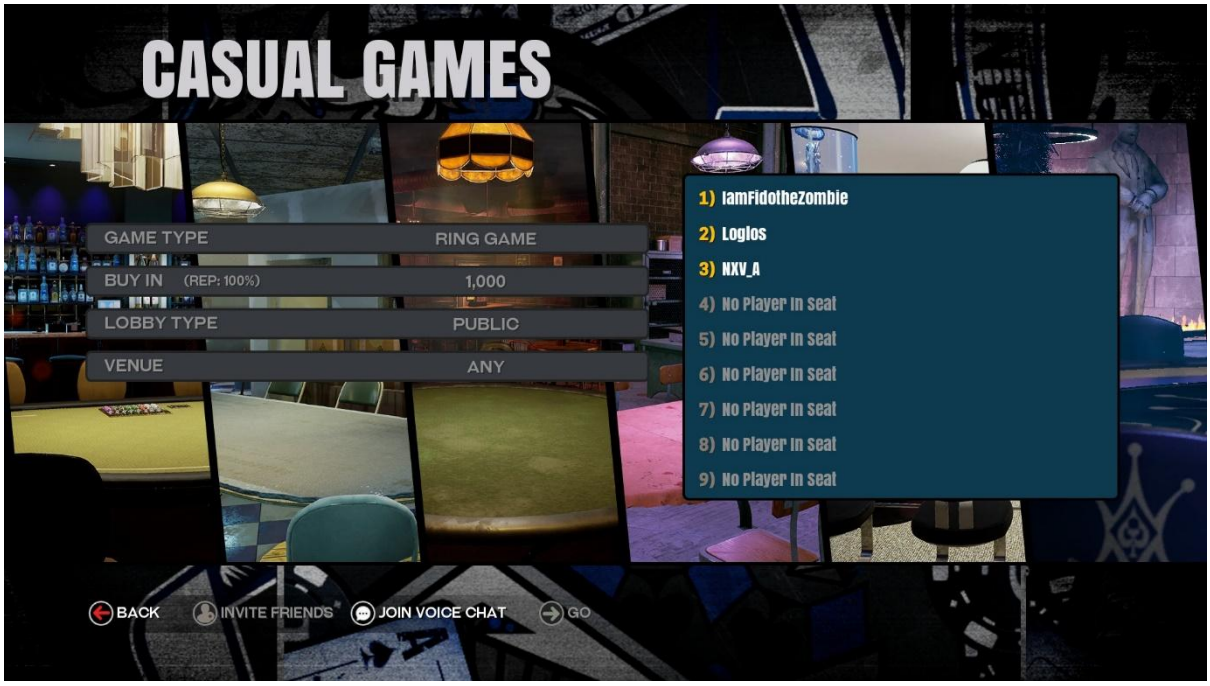
The nickname is a name displayed below the name of the player in game, it serves as a way for player to customise their character further and more uniquely than other options. This

is because by default the player only has one nickname that they can pick and they can unlock others by completing various in-game challenges. This adds a certain level of prestige to players with 'valuable' nicknames and gives a reason for a player to carry on playing.

The cosmetics button allows the player to decorate their player with items they earn or purchase. By default, the user has no means of customising their character as they only are able to use the default clothing. However, if the player plays more, the more chips they will earn and the more items of clothing they will unlock. It being this way adds a sense of uniqueness to each character as well as a level of progression in game.

The head, body, and gender buttons are all self-explanatory. They control the physical features of your character. For example, the head button allows the user to control their head shape as well as the hair and hair colour. All options for physical appearance are available to the user. The body button allows the user to pick their body shape from a list of body shapes as well as skin colour. Lastly, the gender button allows the user to switch their gender between male, or female. Doing this will affect the hair, body shape, and head shape options available to the user.

This menu showcases the players level progression and the rewards the player will receive once attaining a certain level. This gives players another reason to carry on playing, which in turn increases player retention. The rewards given to the player vary from addition chips and cosmetics the user can use to make their character more unique and personal.



This is the multiplayer lobby where players are able to join and play all together in a single game. Here, the player list is visible to the player, this can tell the user who they are playing against and be used to make sure that they have join the correct lobby in the case they are trying to join a friend.

In the lobby, the game settings are made visible to the user. This is useful information to players joining the server so that they know what might happen in the game, and if they want to suggest changes or leave because they don't like the current settings.

The player is also given the option to join the voice chat with other players or invite other friends to the lobby. This helps make the game feel more social able being able to talk in game via their voice and hear the other people and invite their friends to enhance their experience.

Here the game has started, all partaking players are positioned physically around the table and their personalised in-game character is shown to other players.

The chips, cards, and players including the table are physical thing which move around. The chips are thrown in the pot including the cards during dealing. This makes the game feel more immersive making the player feel more involved.

When checking your cards, your character is shown in game to physically check their card further adding to the sense of immersion. The game however does not tell the player their hand, therefore they must figure it out themselves.

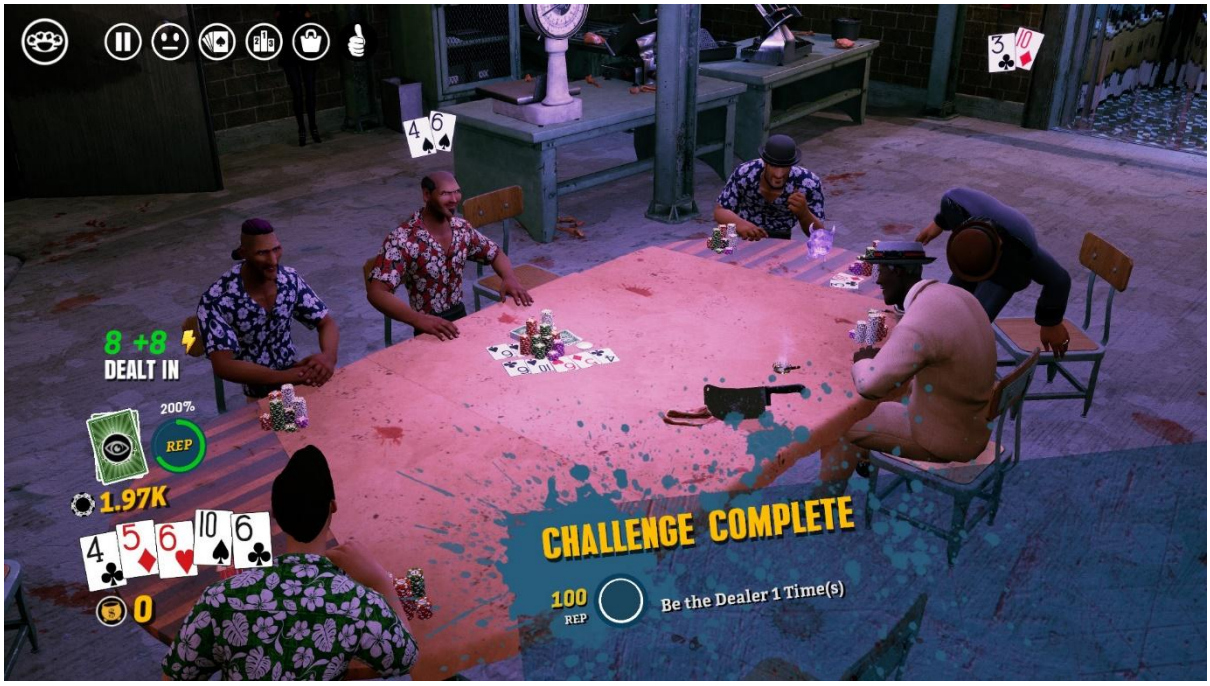
Above each player, a symbol is show representing the last action the player took. For example, one blue arrow represents a call, where as two green arrows represent raising or going all in. The betting amount is displayed above the players name so that it is easy to keep track of the current bet and who is betting what amount. If a player folds a red x is displayed above showing the action they took to the entire lobby.

Additionally, at the start of each round, the role of the players is displayed above the players head where the action would be. This gets removed after the player bets, raises, or folds. The roles are indicated by SB for small blind, BB for big blind, and D for dealer.

The main information of the game is displayed on the left-hand side. This includes the players hand, the players money in-game, the current community cards, and the money in the pot. This is pushed to the side for the reason I'm assuming that game wants the player to focus more on the social aspect of the game by making a majority of the screen filled with the table with the player who have the ability to emote around it.

In the centre the player actions are summarised in a circle with three options, call/check, raise, and fold. These buttons have no key bindings and require the user to use their mouse like most things in the game.

Lastly, the different menus/actions the player can take are shown at the top of the screen. These involve the pause button, the emotes menu, the hand ranking menu, table leader board, and the shop.



Here you are able to see the challenge system in action, and what happens at the end of the game.

The game awarded me with in-game experience points for achieving the goal of a challenge, these challenges are usually quite easy ranging from, call or raise in a game, or be the big blind three times. These can give goals to users giving them a reason to play other than just to play a game of poker. It is also a way for the games level system to work and give you experience points to unlock rewards later on.

When it is time to reveal who has the winning hand. The players who have not folded have their cards displayed above that head whilst the game calculates the winner of the game.

Once the game has figured out who had won the game. The camera of the game focuses on the winner. Whilst this is happening, the game also places effect in the background and overlays GUI indicating the winners name, their hand including the community cards, and the winning sum of money.

This game does not tell you your hand when you look at your cards. Therefore, users must be expected to figure out their hand. This for newer players to poker might be difficult as they wouldn't know what hands there are and what makes them.

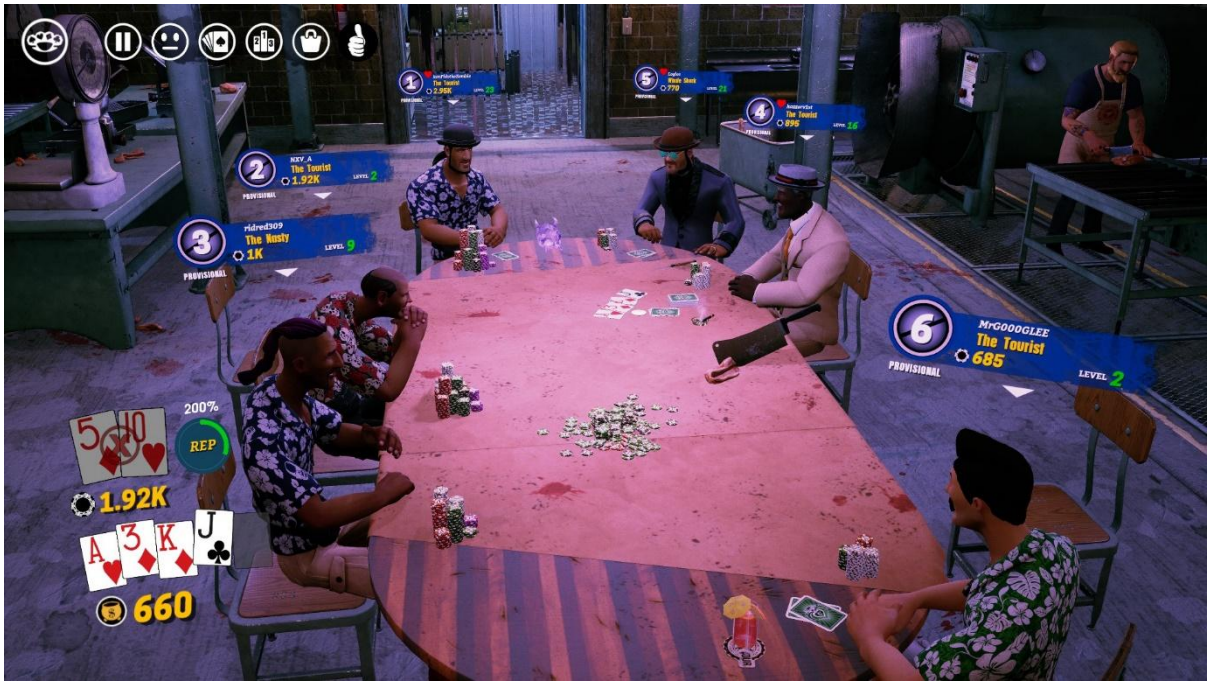
The hand ranking menu shows all the possible hands in poker with the best at the top and the least at the bottom. The game also explains to the user how each hand in made with a little description next to each hand.

The games pause menu offers essential features to the user as well as more game specific ones such as the bounties and daily challenges menu used by players wanting to complete them for their rewards.

The button 'back to game' just returns the player back to the screen they were on – the poker table. The shop and stash are the same as the button in the main menu. The sit out button allows the player to sit out of the game for a period of time until they decide that they would like to play again. This can be useful to player who temporarily need to go do something else but still would like to play.

The rebuy button allows users, if bankrupt to use their overall money to buy themselves back into the game. Help and options is the same as before and the exit button returns the user back to the main menu if they decide to leave

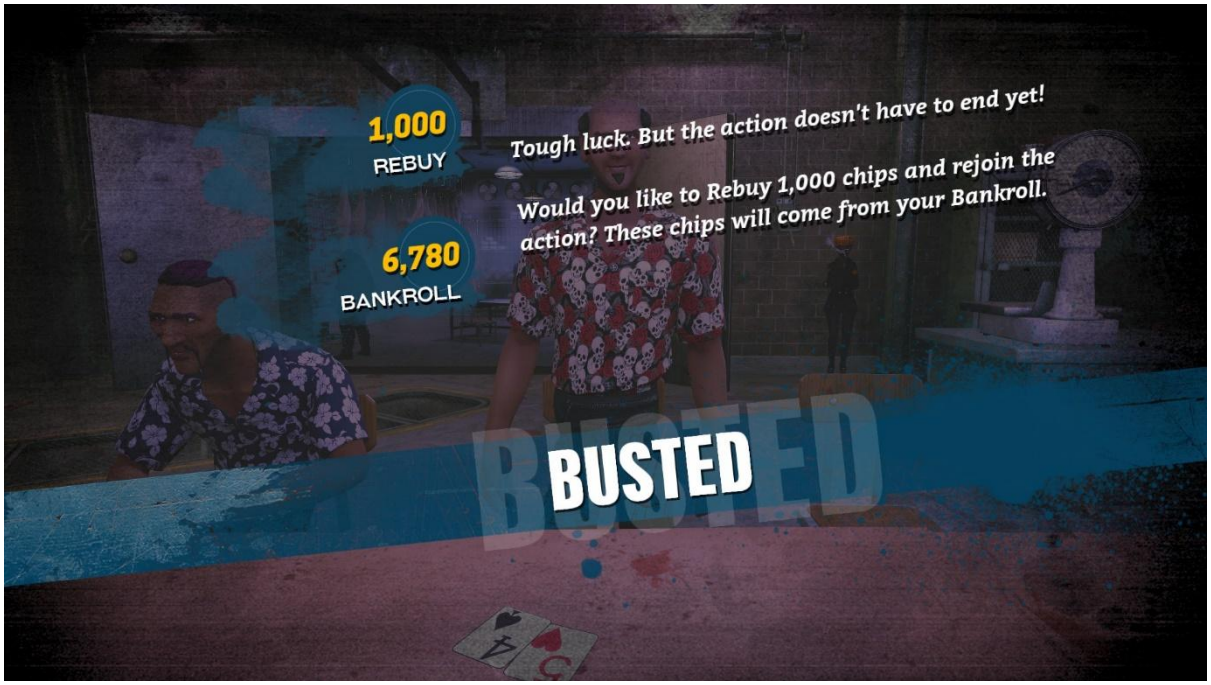
The emote menu is a circular menu consisting of many different types of emotes for players to use to reflect their emotions about a game. For example, if a player wins a large sum, they may want to celebrate it by emoting. When a player emotes, the character will physically act out whatever emote the player chose. The character also makes noises in game depending on the action.



The in-game leader board showcases the player on the table with the amount of chips the player has. The game displays the players name, as well as their chosen nickname below. Next to their name their level is also displayed giving the player more information about the users on their table. Lastly the rank number on the table is displayed on the left. The rank works by lowest equals best. The rank number is determined by the number of chips that the player has – the more chips the better.

Here the user is able to spend their earned in-game money on various cosmetics, in this case table items. These are items which are shown on the table physically and can be interacted with by the player. These are the types of things some player would try to spend a lot of time obtaining just to prove to other that they are better. This helps to increase

replay ability by allowing chips to be used for other things rather than just a number counter.



When a player becomes bankrupt, ‘Busted’ text is displayed on the screen telling the user that they have ran out of money. The game prompts the user if they would like to carry on playing by spending 1000 chips out of their bankroll (‘overall money’) or just quit. If a player decides to rebuy, they will be placed back in the game with 1000 chips again and be waiting to be dealt in on the next turn, otherwise they will be sent back to the main menu where they can decide to quit the game or find a new one.

Objectives

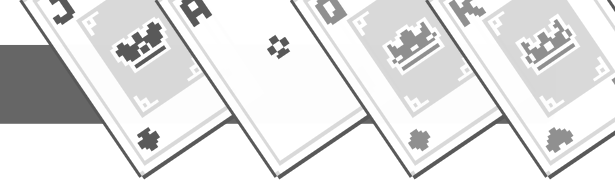
For me to have a good understanding of what my program should end up as, it is beneficial that I write objectives for me to achieve in order to achieve my goal of creating a poker game.

1. Main Menu
    1.1. The user of the program upon opening the game, should be greeted with the main options of the game which include a play button, a settings button, and a quit button.
    1.2. User of the program should be able to select the play button.
        1.2.1. Upon clicking this button, the menu will be updated to show new buttons relating to the button the user just pressed, this being the play button. Users will be asked using buttons whether they would like to play single player or multiplayer or return to the main menu.
            1.2.1.1. Clicking the single player button should start a single player session with 4 other bots who will play along with the player until the user decides to quit.
            1.2.1.2. Clicking the multiplayer button will send the user to a new menu with buttons asking if the user would like to join a server or create one as well as return to the previous menu.
                1.2.1.2.1. Clicking the join lobby button will send the user to another menu where they will be prompted to enter the server ip as well as the server port and then be able to click 'Join' to join that server.
                1.2.1.2.2. Clicking create server will create a server hosting a game for the user and connect that user, then allowing other users to enter the game.
                1.2.1.2.3. Previous button should return the user back to the previous menu.
    1.3. User of the program should be able to select the settings button.
        1.3.1. User will be shown settings that they can change such as username, default port on which servers will be made, full screen, and screen size.
            1.3.1.1. Toggleable settings are a button.
            1.3.1.2. Text boxes, where written inputs are entered, pressing enter will save the information.
    1.4. User of the program should be able to select the quit button.
        1.4.1. Upon clicking select the game should close.

2. Single player
    2.1. If a single player game is made, then the game will fill the other four player slots with bots so that the player is able to play against opponents.
    2.2. Every player starts with the same amount of starting chips ten white chips, five green, three red, two blue, and one black chip. (white = 1, green = 2, red = 4, blue = 8, black = 16)
    2.3. At the start of each round, the player and bots are dealt two unique cards from the deck.
    2.4. Assign a dealer at the start of each round and make the player clockwise to the dealer the little blind and the next player the big blind.
        2.4.1. Make the little blind bet the minimum bet.
        2.4.2. Make the big blind bet double the little blind.

2.5. The player which would play first in a round would be the player next to the big blind.
2.6. Check if the current player has folded, if so, skip their go.
2.6.1. If the player hasn't folded prompt them to make their go.
2.6.1.1. Players and bots should be able to fold, and bet.
2.7. If everyone has bet the same the first three community card are then revealed
2.7.1. The players are then looped over again, if everyone bet the same then the 4th community card is revealed.
2.7.2. The players are then looped over again, if everyone bet the same then the 5th and final card is revealed.
2.7.3. The showdown is then played until everyone bets the same.
2.7.3.1. The player with the highest value hand wins the pot or is split among the joint highest hands.
2.8. If every player but one has folded, then the last player wins the pot regardless.
2.9. After the winner has received their winnings, a short intermission will occur, afterward a new round will be played.
2.10. If the player quits, by either closing the game or returning to the main menu, the game will end.

3. Multiplayer
3.1. Similar to single player but allows other players to be able to join and play in the same game.
3.2. Empty player slots are filled with bots.
3.2.1. If a player joins mid game, the game will wait for the round to end before removing a bot and adding the player to the game.
3.3. If the game is full of players, then the user trying to connect to server will not be connected to server.
3.4. Every player starts with the same amount of starting chips ten white chips, five green, three red, two blue, and one black chip.
3.5. At the start of each round, the player and bots are dealt two unique cards from the deck.
3.6. Assign a dealer at the start of each round and make the player clockwise to the dealer the little blind and the next player the big blind.
3.6.1. Make the little blind bet the minimum bet.
3.6.2. Make the big blind bet double the little blind.
3.7. The player which would play first in a round would be the player next to the big blind.
3.8. Check if the current player has folded, if so, skip their go.
3.8.1. If the player hasn't folded prompt them to make their go.
3.8.1.1. Players and bots should be able to fold, and bet.
3.9. If everyone has bet the same the first three community card are then revealed
3.9.1. The players are then looped over again, if everyone bet the same then the 4th community card is revealed.
3.9.2. The players are then looped over again, if everyone bet the same then the 5th and final card is revealed.
3.9.3. The showdown is then played until everyone bets the same.



3.9.3.1. The player with the highest value hand wins the pot or is split among the joint highest hands.

3.10. If every player but one has folded, then the last player wins the pot regardless.

3.11. After the winner has received their winnings, a short intermission will occur, afterward a new round will be played.

3.12. If a player quits, by either closing the game or returning to the main menu, the game will end for the user that left.

3.12.1. If they are the host, the server will close.

3.13. The players in the game will be able to communicate with one another via an in-game chat.

4. In-Game Character

4.1. Allow the user, under the settings menu, to be able to change their name and profile picture.

4.1.1. Placing an image in the 'User Images' folder and then specifying the name of the image in game under the settings.

4.1.1.1. If no image is selected or is unable to be loaded, a default one is chosen.

4.1.2. Allow users to change their in-game name (maximum of 30 characters)

4.2. During gameplay allow the user to increase and decrease their bet, check and fold.

4.3. Allow the user to peek at their hand when holding down the spacebar.

4.4. In game the user's chips should be drawn on the screen so that the player knows their chips

4.5. The users profile picture and username should be drawn on the screen so that they know what they look like to other players.

5. In-game UI

5.1. Show the player the current community cards at the top of the screen.

5.2. Show the player the other bots/players partaking in the match.

5.2.1. Each individual player's chips are drawn for the player.

5.2.2. Show the current status of a player.

5.2.2.1. If a player folds, text is displayed that they did so.

5.2.2.2. The last action is shown, for example, check, call £5, raise £7

5.2.2.3. The current role of the player is displayed next to their displayed details with an icon. Showing whether they are a dealer, little blind, or big blind.

5.3. Show the player the controls on the right side of the screen.

5.3.1. Have text displayed alongside the button shown so that the user knows what the key controls what in game.

5.3.2. When the player increases or decreases their bet, it should be visible next to the betting controls.

5.4. Show the game information at the top of the screen.

5.4.1. Showcase the time that game has been running for.

5.4.1.1. Show the game number and the turn number.

5.4.2. Display the name of the player who's go it is currently.

6. Game Logic
   - 6.1. During gameplay, be able to cycle between the main stages of the game: the pre-flop, the flop, the turn, the river, and the showdown.
     - 6.1.1. The 'round' is only progressed forward once everyone playing in that round has bet the same amount.
       - 6.1.1.1. The pre-flop is the first round, no community cards are shown.
       - 6.1.1.2. The flop is the second round, three community cards are shown.
       - 6.1.1.3. The turn is the third round where fourth card is revealed.
       - 6.1.1.4. The river is the fourth round where the final community card is shown.
       - 6.1.1.5. The showdown is the final round where players remain betting until they all bet the same.
         - 6.1.1.5.1. The player with the highest value hand wins the pot.
         - 6.1.1.5.2. Joint winners will have the pot split between winners.
     - 6.1.2. Game is ended if all but one player folds.
       - 6.1.2.1. Last remaining player wins the pot regardless.
     - 6.1.3. Game will remain continuous unless the user decides to leave.

7. Bots
   - 7.1. Have a unique name and associated profile picture shown to other players.
   - 7.2. Able to bet and fold like other players.
     - 7.2.1. Decisions based on hand ranking, amount of money, money in the pot and an overall 'confidence level' assigned to each bot between 0-100 but capped at 10-90 so that the bot isn't overly aggressive or passive.
   - 7.3. Take a variable amount of time to make their decision so that it feels more natural to players instead of the choice being made instantly.

8. Poker Assistance
   - 8.1. Allow players to look at a table of hands during gameplay so that they can see what hands are possible and their ranking with percentage chance to obtain that hand.
   - 8.2. A brief description on the rules and gameplay of Texas hold-em poker.
   - 8.3. Table provided in game showing starting hand winning percentages to help newer players decide on how they would like to play with their hand.

9. Game Settings
   - 9.1. Allow the user to change a game setting which controls their username.
   - 9.2. Allow the user to change a game setting which controls which profile image is loaded when their character is loaded.
   - 9.3. Allow the user to enable and disable full screen setting.
   - 9.4. Allow the user to change the setting which controls the dimensions of the screen.

10. Art Style & Sound Design
    - 10.1. The game being called pixel poker means that the in-game assets and images should follow a pixel art style.

10.1.1. As the user images can be chosen by individual players, these don't have to be in a pixel art style.

10.2. Minimal sounds will be used in game.

10.2.1. When pressing a button a sound should be played.

10.2.2. When increasing or decreasing your bet, a sound should be played.

10.2.3. When in the main menu, main menu music should be played.

10.2.4. When playing the game, music should be played.

11. Communication

11.1. Text chat in game to be able to communicate with other players.

11.1.1. Ability to send text messages to other players within a multi-player game.

11.1.2. Should support Unicode characters.

11.1.3. Players are only able to type in the chat if they type in the message box.

11.1.3.1. Message is only sent when the enter key is pressed.

11.1.4. Messages from other player should be shown in the chat.

11.1.5. Close or open chat.

12. Save Data

12.1. Save user settings under a text file.

12.1.1. Data being username and profile picture plus settings.

12.2. When opening the game, the settings should be loaded.

12.2.1. Using json.loads() to load the settings into the game.

12.3. If there is no settings file, then create a new default one.

Objective Reasoning

Now that I have chosen my objectives. I must now explain the reasoning behind the need of the objectives that I have set for my program to be there and why they are the way they are. Additionally, I will explain more about each objective.

Main menu:

The main menu is a key feature needed for users of my program to navigate it and find what they want to do, for example start a single player game or create a multiplayer lobby. It should be easy to follow, use, and understand with buttons and text labels informing the user of its function. Not only this, but the button will also need to provide the function expected of it, for example – ability to close the game, open the settings so that users are able to customise their character and change individual settings, and chose whether they would like to play single player or multiplayer.

The menu doesn't need to be overly complicated only containing the buttons needed for the user to navigate the program. Additionally, it doesn't need to have any graphical design as due to time constraints I won't be able to focus on non-beneficial features.

Single player:

When a user creates a single player game the player and the four other bots should start with a set amount of chips, for example 10 white chips, 5 green chips, 3 red chips, 2 blue chips, and 1 black chip. The player should be able to see the bots clearly with their information in the bottom middle of the screen.

The players character information such as username are loaded upon creation, and then displayed in the bottom centre of the screen.

The game information such as time, community cards, and pot should be displayed in a way where they are easily seen by the player and don't intrude on gameplay.

The bots in the game should feel fun to play against and take varying amounts of time to take their action so that the game feels more fluid rather than the bot making their decision instantly.

The game should run on standard Texas hold-em rules.

Multiplayer:

Multiplayer should work broadly the same as single player. The major but most important difference being the fact that other players would be able to connect to the server and all play in the same game. The game should display the other players correctly on screen relative to the player. Therefore, the list would need to be normalised for the user so that their client is able to draw the information regarding the players on the screen.

The information across clients must all be the same therefore the information sent to the individual players regarding the game must be sent at the same time.

Player will be able to either create a server or join one by entering its Ip address and server port and then connecting to the server where they will receive information and be able to play on the server. If not all player slots are filled, then bots will fill these places.

In-game character:

The in-game character is how other players, and the main user will see themselves in a game of poker. The user will be able to set their in-game name to a maximum limit of 30

characters, and set their profile picture to any image they want. If no image or name is set, then defaults would be applied. This allows unique players and the ability of personalisation, especially when considering multiplayer.

Bots:

The bots shouldn't all play the same or play predictably. However, depending on the bot and other factors such as hand value and their money, they should play slightly differently. For example, when a bot is created a value could be picked up saying how passive or aggressive a bot will play.

The bots should be able to do all the actions a player can including, check, call, raise, and fold.

Poker assistance:

In game there should be a menu which would be openable with a key press for example H or Tab, which shows the user all the possible hands in poker with best being at the top and worst at the bottom. This can help newer players gain a better understanding of poker as well as the strength of their hand. The menu could also provide an explanation of the rules of the game as well as a chart explaining the starting poker hands winning probabilities.

Game settings:

Game settings should be simple and consist primarily of in game username, profile picture, and screen resolution. These should be remembered every time the game is opened if the user applies the changes.

The game settings should be simple and easy to understand. The user should be able to be able to easily change the settings with minimal fuss. Settings which could be controlled by the user are username, profile picture, screen settings such as size.

The game store these settings and then should load these settings every time upon start up. If there isn't a settings file, then one would be created for you.

Art style & sound design:

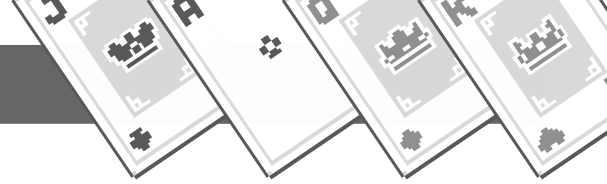
The art style is pixel based so most text in game and graphical art etc should follow that style otherwise there would be a continuity error with how the UI looks. However, since the player profile pictures are able to be customised per user, it would be hard to follow the pixel art style all the time. The sound design doesn't need to be anything special, just audio with complements the user's actions and what is happening in game.

Communication:

An in-game chat should exist so that players are able to communicate with each other during a match of poker. The chat could be a box in the corner which could be toggled by the player and that supports Unicode characters.

Save Data:

The save data doesn't need to store anything major just important info such as your username, profile picture, and any other setting you may or may not have changed. This could be a text file stored along with the program. If the program does not have a settings/save data file, one could be created automatically for the user.



Game System

I will make my game system take inspiration from both the Red Dead Redemption 2 poker minigame and Prominence Poker. I want to take the more serious aesthetic and easier playing environment of poker from RDR2 and combine it with some social features like Prominence Poker. I hope to also incorporate a pixel art style in my game and borrow certain GUI elements and placements. The game will include a main menu screen where players are able to join single player or multiplayer games, create multiplayer servers, change settings, and quit. Players in single player games should be able to play against bots which make decisions based on their hand and money. Multiplayer games should allow players to play against each other in the same game. The game should autofill empty spaces with bots.

Time Constraints

Due to the time available to be, and the fact that I am a single person and not an entire development studio with lot of resources and employees. I will be unable to create a fully featured poker game with all the bell and whistles I would like to add. For example, spending time on the art on this game isn't strictly necessary or required – and does not provide me with any additional marks. Additionally, adding extra features such as a full tutorial or a VOIP (Voice over Ip) chat isn't needed for a poker game, but would add more depth to the overall game. However, despite this, I will still add the main core gameplay as well as additional features such as poker assistance.

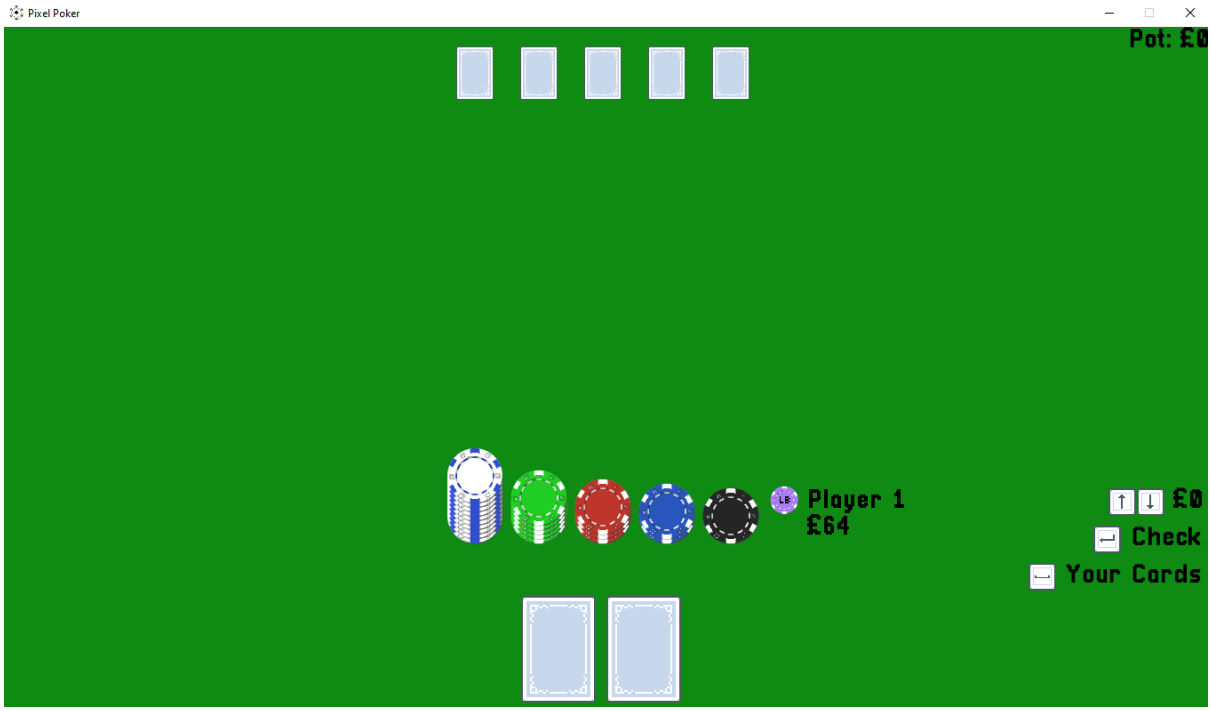
Design

Prototype

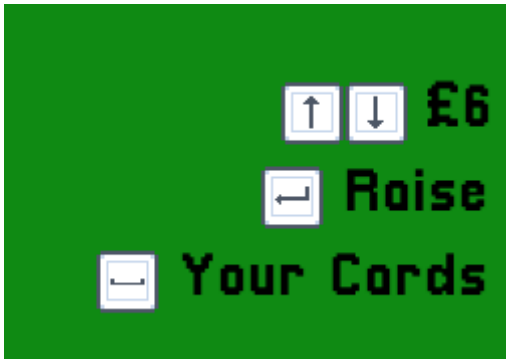
Reasoning

To help aid the design of my program, I decided to make a quick basic prototype involving the basic algorithms which might be needed on my final product. This prototype was helpful as it was a good way to see what the final game might look like without having to make it fully yet acting as a proof of concept. Due to this, it allowed me to change parts of the design that I personally didn't find good before making the program. This also allowed me to see what my clients thought of my design and suggest changes before I came fully committed to the design. Having an idea of what the program should look and feel like gave a good driving force during the development.

Overall



The prototype only has this screen, however despite this, it gave me and my clients a good idea of what the main gameplay screen would look like, however. The UI elements are placed in a logical way where they can be easily seen by the user. The main elements of the GUI are separated in logical sections which include they players information in the bottom middle of the screen, the community card at the top middle, the controls in the bottom right corner, and the pot chips in the centre with the total in the top right.

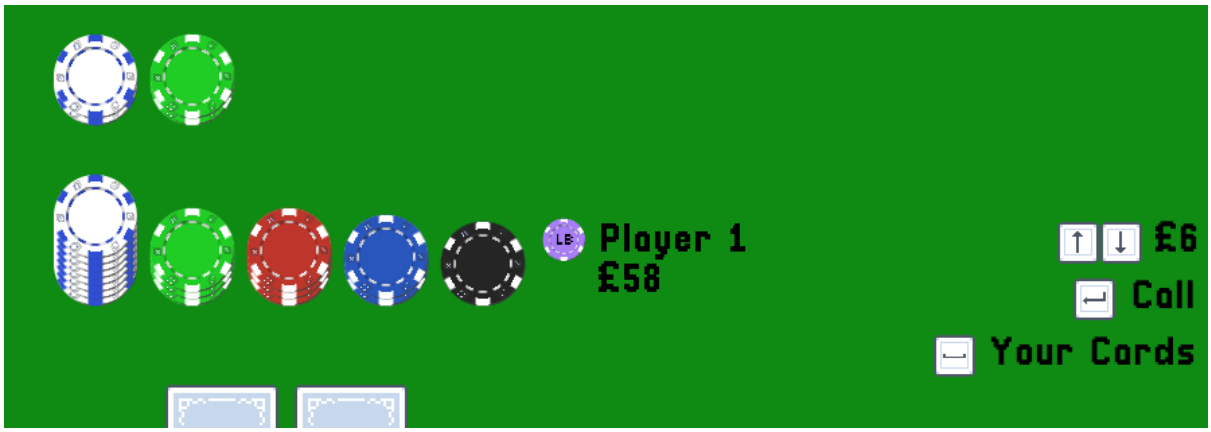


These controls easily tell the player what button on their keyboard controls what with easy to see icons representing the button that they control. These controls being in the bottom right corner allow the controls to be out of the way not obscuring gameplay, whilst also being always on the screen so if a player forgets the controls they can easily be reminded by just looking in the bottom right.

The betting action name automatically updates depending on the action that the user will take, for example as seen in the first image, if the current bet is zero, the text is 'Check'. However, if the user decides to raise the bet, the text is then replaced with 'Raise'.

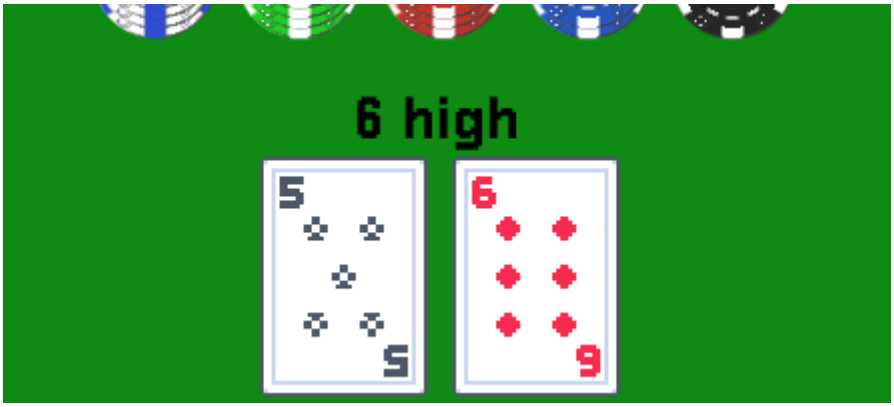
Having the raise and lower bet controls on the arrow keys made sense as other games use those keys to increase and decrease values, and therefore would be intuitive to the player. This also makes sense due to the fact that the arrows correspond with the action, the up arrow increase the bet, and the down arrow decreases it.

Having the players hand not being revealed all the time is a strange decision, as most other poker games have your hand constantly on the screen so that you don't forget what you have. However, I found that along with my clients, making it so that the user if they would like to look at their hand, must actively do so. In my case press and hold the space bar. This made it so that despite it being a program on the computer, it made the player feel more involved and actually part of the game rather than being a passive observer by constantly checking their cards to see what their hand is as if it was a poker game in the real world.



Here, once the player has bet the amount they have chosen, the money is deducted from the player and added to the pot, where the counter for the amount of money in the pot increases. The change of money is also visible on the screen where the chips are visibly taken from the player and then draw in the middle of the screen where the pot is. This also causes the betting control to now display 'Call' where it once displayed 'Raise' to indicate to the player that that is the current betting amount. Additionally, the betting controls don't allow the user to bet more than they have or lower than the amount they have or the current betting amount.

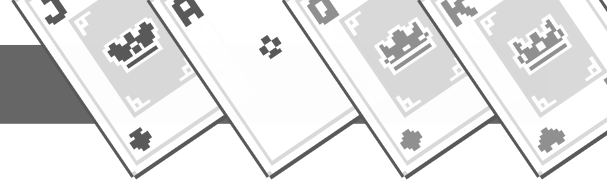
The player information also shows the money of the player, the username – in this case 'Player 1', and the role of the player – the 'Little Blind'. This information is useful to the player as knowing how they look like to other players as well as their role in the game with the number of credits they have can help inform decisions of the player. These include, should I change my in-game name? Should I raise or call?



When the user is actively looking at their hand in the prototype, the value of the hand is actively displayed above it, this form of poker assistance is useful to both new players of poker and old ones. This allows newer players who don't know all the hands and how they are formed to assess their chance of winning and make decisions informed on their hand value. It also allows older player, who might know what the hand are and what form them to quickly play rather than just sit and figure out the value of their hand every time.

The cards being displayed at the top of the screen is intuitive and when talking to my clients about this, they like them being in the centre, but not intrusive compared to rest of the screen and elements. The community cards being drawn in the middle of the screen is good as most users would be focused on the centre of the screen, therefore being intuitive and easy to see compared to being drawn in the corner or off to the side. Instead of the cards being front and centre, the cards are pushed towards the top of the screen. This is because usually players are more concerned about the pot which is drawn in the centre of the screen where most players are looking.

Five cards are always shown even if no community cards are revealed to the player. This indicates to the player that a maximum of five cards would ever been shown at one time and during gameplay it is easy to see how many cards are left remaining to be revealed. This helps newer players understand how many community cards are meant to be drawn, and how many are left to be added.



Prototype Code

```
import random, pygame, os, sys

pygame.init()
pygame.font.init()
size = width, height = 1360, 768
screen = pygame.display.set_mode(size) #, pygame.FULLSCREEN |
pygame.DOUBLEBUF )
FPS = 60
clock = pygame.time.Clock()
getImg = pygame.image.load
pygame.display.set_caption('Pixel Poker')
icon = getImg(os.path.join("Assets/Chips/", "Poker icon.png"))
pygame.display.set_icon(icon)
midX = size[0] // 2
midY = size[1] // 2
my_font = pygame.font.Font(os.path.join("Assets/Fonts/","Pixellari.ttf"),
30)

textnames = {
    "11": "Jack",
    "12": "Queen",
    "13": "King",
    "14": "Ace",
}

playersNo = 5
cardsPack = {}
defaultChips = [10, 5, 3, 2, 1]
playerChips  = [defaultChips]* playersNo

for type in ["Spades", "Diamonds", "Hearts", "Clubs"]:
    for i in range(2, 14):
        name = str(i)
        try:
            name = textnames[name]
        except:
            pass
        cardsPack[name + " of " + type] = i
    cardsPack["Ace of " + type] = 14

players = []
for i in range(playersNo):
    players.append([])

tempCards = list(cardsPack.items())

for player in players:
    for i in range(2):
        index = random.randint(0, len(tempCards) - 1)
        card = tempCards[index]
        tempCards.pop(index)
        player.append(card)


playerRole = 'Little blind'
player = 1
currentBet = 0
pot = [0, 0, 0, 0, 0]
community = [[], 0]
```

```
def cardType(value):
    if value == 11:
        return("Jack")
    elif value == 12:
        return("Queen")
    elif value == 13:
        return("King")
    elif value == 14:
        return("Ace")
    elif value < 11 and value > 1:
        return str(value)
    else:
        raise TypeError("Integers 2-14 are only supported")


def handCheck():
    bestCard = 0
    playerCom = []
    playerCom.append(players[player][0])
    playerCom.append(players[player][1])
    for i in range(community[1]):
        playerCom.append(community[0][i])
    if players[player][0][1] == players[player][1][1]:
        return("Pair of " + cardType(players[player][0][1]) + "'s")
    elif players[player][0][1] > players[player][1][1]:
        bestCard = 0
    elif players[player][1][1] > players[player][0][1]:
        bestCard = 1
    return(cardType(players[player][bestCard][1]) + " high")


def handLook(keys_pressed):
    crop = (22, 4, 84, 120)
    if keys_pressed[pygame.K_SPACE]:
        screen.blit(card1, (midX - 96, size[1] - 128), crop)
        screen.blit(card2, (midX, size[1] - 128), crop)
        text_surface = my_font.render(handCheck(), True, (0, 0, 0))
        text_size = my_font.size(handCheck())
        screen.blit(text_surface, (midX - (text_size[0] / 2) - 7, size[1] -
160))
    else:
        screen.blit(back, (midX - 96, size[1] - 128), crop)
        screen.blit(back, (midX, size[1] - 128), crop)


def drawChips():
    spacingX, spacingY = 72, 5
    chipX, chipY = midX - (spacingX * len(playerChips[player]) // 2),
size[1] - 250
    startY = chipY
    count = 1
    for index in range(len(playerChips[player])):
        for i in range(playerChips[player][index]):
            screen.blit(getImg(os.path.join("Assets/Chips/", str(count) +
".png")), (chipX, chipY))
            chipY -= spacingY
        chipX, chipY = chipX + spacingX, startY
        count = count * 2
```

```
def bet(amount):
    global currentBet
    global pot
    currentBet = amount
    if amount == 0:
        return "Check"
    while amount > 0:
        decreased = False
        for i in range(4, -1, -1):
            if ((amount % (2 ** i)) == 0) and amount > 0 and
playerChips[player][i] > 0:
                playerChips[player][i] -= 1
                pot[i] += 1
                amount -= 2 ** i
                decreased = True
        if not decreased and amount > 0:
            if playerChips[player][0] > 0:
                playerChips[player][0] -= 1
                pot[0] += 1
                amount -= 1
            else:
                return 'Impossible'
            playerChips[player][0] -= 1
            pot[0] -= 1
            amount -= 1


possible = []
def possibleValues():
    global possible
    possible = []
    for i in range(1, 5):
        duplicate = playerChips[player]
        looping = True
        while looping:
            looping = False
            for k in range(0, 5):
                if (duplicate[k] >= 0) and (i != k):
                    possible.append(totalMoney(duplicate))
                    duplicate[k] -= 1
                    looping = True


def totalMoney(chips):
    total = 0
    count = 1
    for index in range(len(chips)):
        total += chips[index] * count
        count = count * 2
    return total


def drawPlayerInfo():
    posX, posY = midX + (midX // 3), size[1] - 250
    Name = my_font.render("Player " + str(player), False, (0, 0, 0))
    Money = my_font.render("£" + str(totalMoney(playerChips[player])),
False, (0, 0, 0))
    if playerRole != '':
```

```
screen.blit(pygame.transform.scale(getImg(os.path.join("Assets/Chips/",
playerRole + ".png")), (32, 32)), (posX- 42, posY - 2))
    screen.blit(Name, (posX, posY))
    screen.blit(Money, (posX, posY + 30))


betting = currentBet
def drawControls():
    global betting
    posX, posY = size[0] - 125, size[1] - 250
    maxX, spacing, spaceY = 100, 0, 42
    up, down = pygame.transform.scale(getImg(os.path.join("Assets/Keys/",
"Up.png")), (32, 32)),
pygame.transform.scale(getImg(os.path.join("Assets/Keys/", "Down.png")),
(32, 32))
    enter = pygame.transform.scale(getImg(os.path.join("Assets/Keys/",
"Enter.png")), (32, 32))
    space = pygame.transform.scale(getImg(os.path.join("Assets/Keys/",
"Space.png")), (32, 32))
    if betting > currentBet:
        option = "Raise"
    elif betting == currentBet and currentBet != 0:
        option = 'Call'
    elif currentBet == 0:
        option = 'Check'
    textSize = my_font.size(option)
    betSize = my_font.size(str(betting))
    betTxt = my_font.render("£" + str(betting), True, (0, 0, 0))
    showTxt = my_font.render("Your Cards", True, (0, 0, 0))
    showSize = my_font.size("Your Cards")
    text = my_font.render(option, True, (0,0,0))
    spacing = (betSize[0] + (posX + spaceY)) - (maxX + posX)
    screen.blit(up, (posX - 32 - spacing, posY))
    screen.blit(down, (posX - spacing, posY))
    screen.blit(betTxt, (posX + spaceY - spacing, posY))
    spacing = ((textSize[0] + (posX + spaceY)) - (maxX + posX)) - 15
    screen.blit(enter, (posX - spacing, posY + spaceY))
    screen.blit(text, (posX + spaceY - spacing, posY + spaceY))
    spacing = ((showSize[0] + (posX + spaceY)) - (maxX + posX)) - 15
    screen.blit(space, (posX - spacing, posY + 84))
    screen.blit(showTxt, (posX + spaceY - spacing, posY + 84))


def drawPot():
    spacingX, spacingY = 72, 5
    chipX, chipY = midX - (spacingX * len(pot) // 2), midY
    startY = chipY
    count = 1
    potSize =  my_font.size("Pot: £"+ str(totalMoney(pot)))
    potTxt = my_font.render("Pot: £"+ str(totalMoney(pot)), True, (0, 0,
0))
    screen.blit(potTxt, (size[0] - potSize[0], 0))
    for index in range(len(pot)):
        for i in range(pot[index]):
            screen.blit(getImg(os.path.join("Assets/Chips/", str(count) +
".png")), (chipX, chipY))
            chipY -= spacingY
        chipX, chipY = chipX + spacingX, startY
        count = count * 2
```

```
def getCommunity():
    for i in range(5):
        index = random.randint(0, len(tempCards) - 1)
        card = tempCards[index]
        tempCards.pop(index)
        community[0].append(card)


calledCommunity = 0
def drawCommunity(update):
    global calledCommunity
    if calledCommunity == 1:
        community[1] = 3
    elif calledCommunity == 2:
        community[1] = 4
    elif calledCommunity == 3:
        community[1] = 5
    spacingX, spacingY = 72, 5
    cardX, cardY = midX - (spacingX * len(community[0]) // 2), 20
    for index in range(community[1]):
        screen.blit(getImg(os.path.join("Assets/Cards/",
community[0][index][0] + ".png")), (cardX, cardY))
        cardX += spacingX
    for index in range(len(community[0]) - community[1]):
        screen.blit(getImg(os.path.join("Assets/Cards/", "card_back.png")),
(cardX, cardY))
        cardX += spacingX
    if update:
        calledCommunity += 1


card1, card2 = getImg(os.path.join("Assets/Cards/", players[player][0][0] +
".png")), getImg(os.path.join("Assets/Cards/", players[player][1][0] +
".png"))
back = getImg(os.path.join("Assets/Cards/", "card_back.png"))
card1, card2, back = pygame.transform.scale(card1, (128, 128)),
pygame.transform.scale(card2, (128, 128)), pygame.transform.scale(back,
(128, 128))

green = (15, 138, 19)
getCommunity()


Running = True
while Running:
    clock.tick(FPS)
    screen.fill(green)
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            Running = False
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_ESCAPE:
                Running = False
            elif event.key == pygame.K_UP and
totalMoney(playerChips[player]) > betting:
                betting += 1
            elif event.key == pygame.K_DOWN and currentBet < betting:
                betting -= 1
            elif event.key == pygame.K_RETURN:
                bet(betting)
```

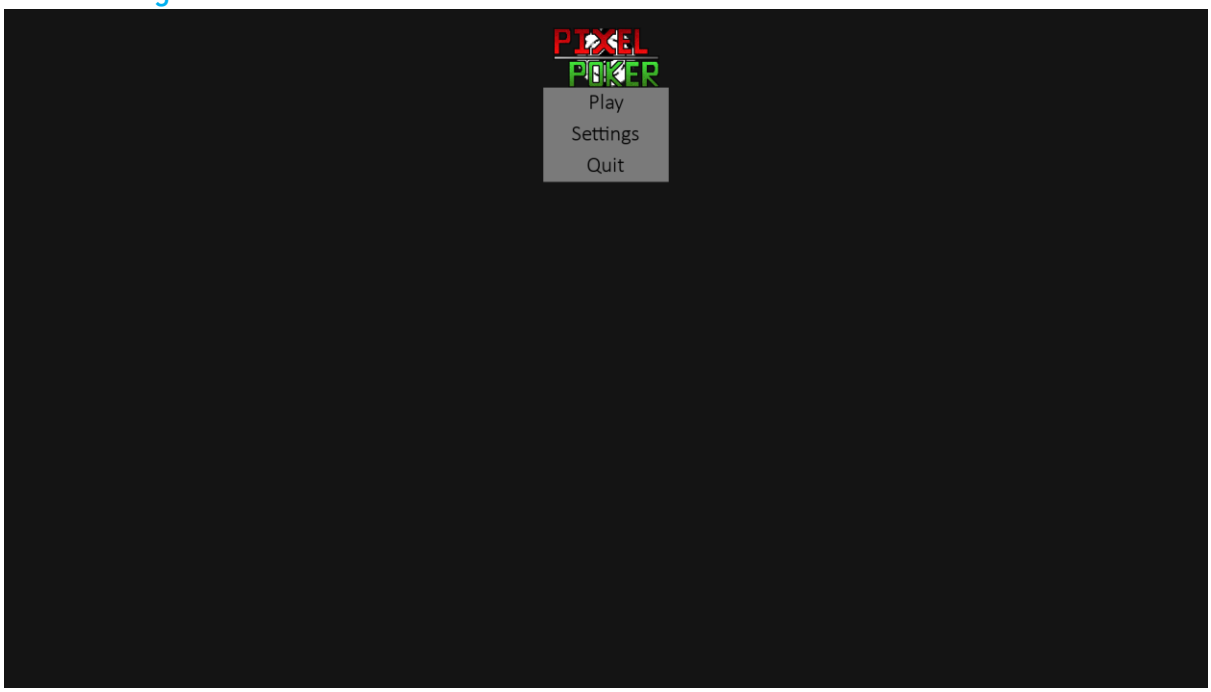
```
            elif event.key == pygame.K_c:
                drawCommunity(True)
    drawControls()
    keys_pressed = pygame.key.get_pressed()
    handLook(keys_pressed)
    drawChips()
    drawPot()
    drawCommunity(None)
    drawPlayerInfo()
    pygame.display.update()

pygame.quit()
exit()
```

Menus

Reason and need for menus

Menus in any game are a key feature, it allows users of the program to navigate and traverse the program to get to the part that they want. In my case allowing the user to change settings, play the game, and also quit the game.

Menu Design

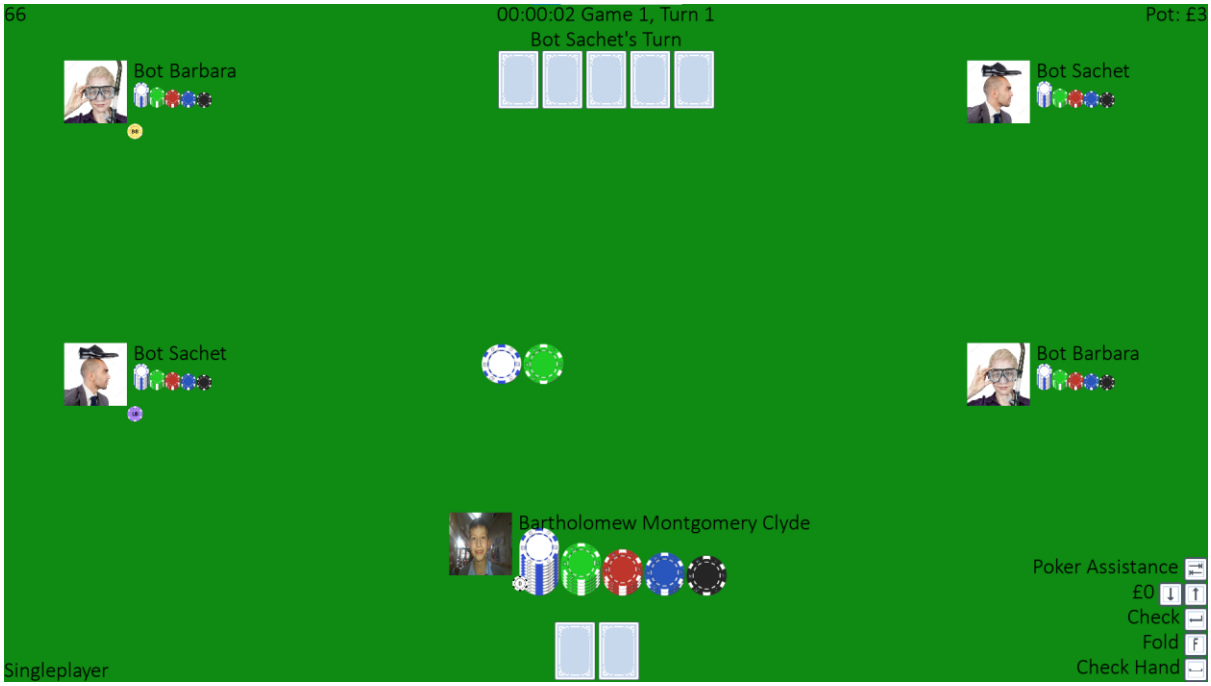


My menu design is pretty basic as I did not want to waste time on the style and artwork for the main menu as I stated due to time constraints.

The menu showcases the logo of the game at the top and three very simple options for the user to choose from; play, settings, or quit. Upon clicking these buttons, the user would be transported to the corresponding menu.

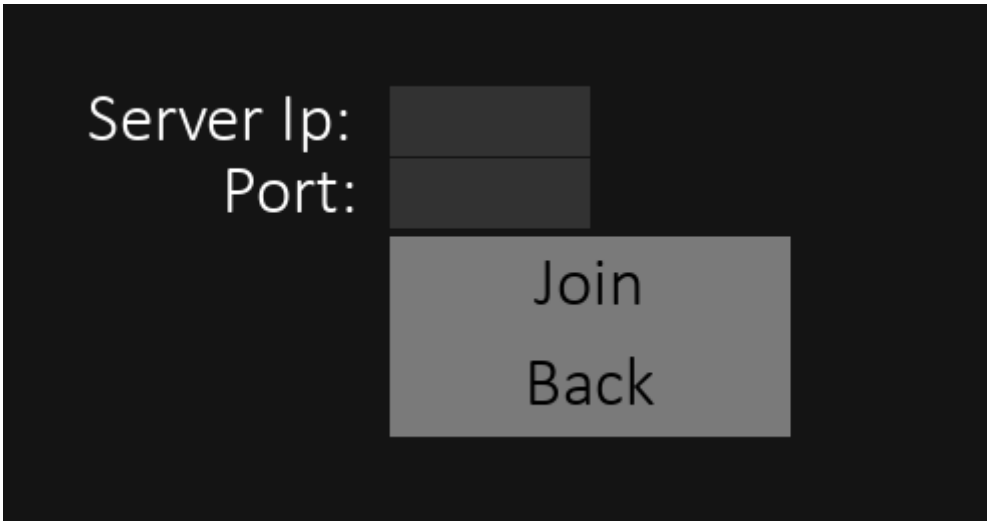
When the user hover over the button with their mouse cursor, the button will react by changing colour to signify the fact that the user's cursor is colliding with button. If the user were to click on the button, it would briefly change the colour to a different colour to show the user that the click had been registered and then run the function assigned to the button. In the case of the play button, it sends the users to the menu asking whether they would like to play singleplayer or multiplayer.

On this new menu, the user can specify whether they would like to play a singleplayer or multiplayer game. If the user is to select the singleplayer option, then it would send the user into a singleplayer lobby where the other four players are bots.



Here the user is sent to a singleplayer lobby where they can play against bots until the player decides to quit.

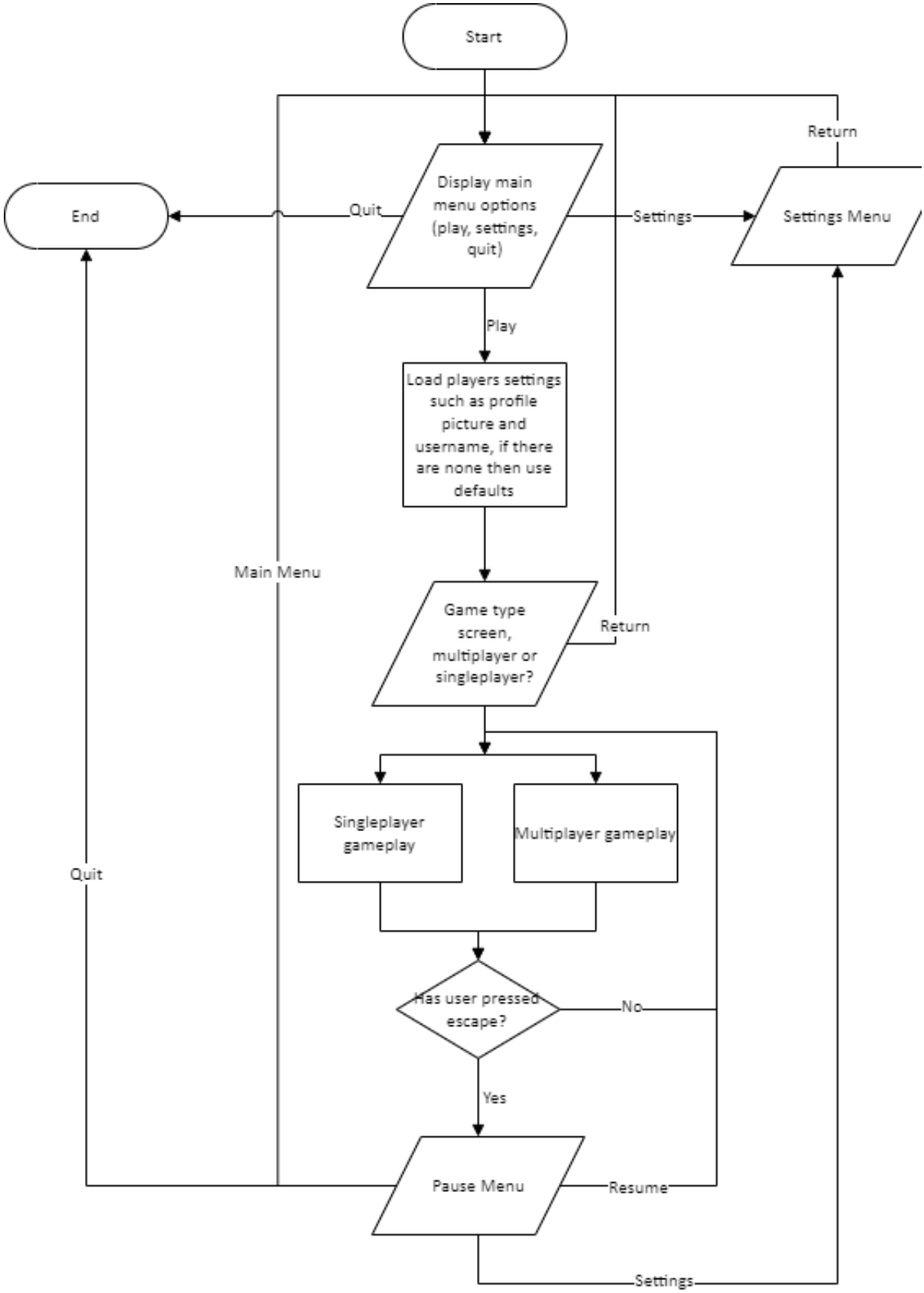
If the user is to instead select the multiplayer option, they are presented with two buttons. One where the user is able to host and create a multiplayer lobby in which other users are able to join, and a joining an ongoing multiplayer lobby.



Here the user can type in the text boxes provided. The top box will take the servers Ip address, and the bottom will take the server port. If the server Ip and or server port that the user provides is incorrect or is unable to connect, the game will be sent back to the main menu.

If the information provided in the boxes is correct and allows the user to connect to the server. The game will be loaded on the clients end and will be loaded into the game.

Menu Flowchart



UI Elements

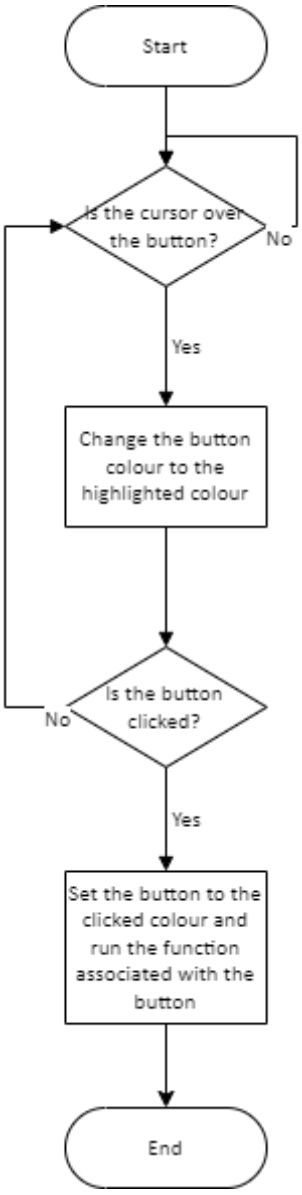
Button

To make my design process easier when it comes to GUIs, creating a button class would be essential as buttons are a major part of any graphical user interface. Making the buttons a class allows me to have multiple instances of buttons at one time whilst also making each one unique. For example, varied size, colour, and text.

The buttons colour will update to a brighter colour when the mouse cursor is hovering over the button. The colour is automatically generated from the base colour by editing the RGB values. If the mouse cursor collides with the button's collision box, it returns true, which then allows the button colour to be updated to the highlighted state.

If the button is then clicked and the mouse cursor is colliding with the button. Then the button is then set to the clicked colour which is dark and is also automatically generated from the base colour. If there is a function associated with the button, then the function will be run, if not, it would be ignored.

Button Flowchart



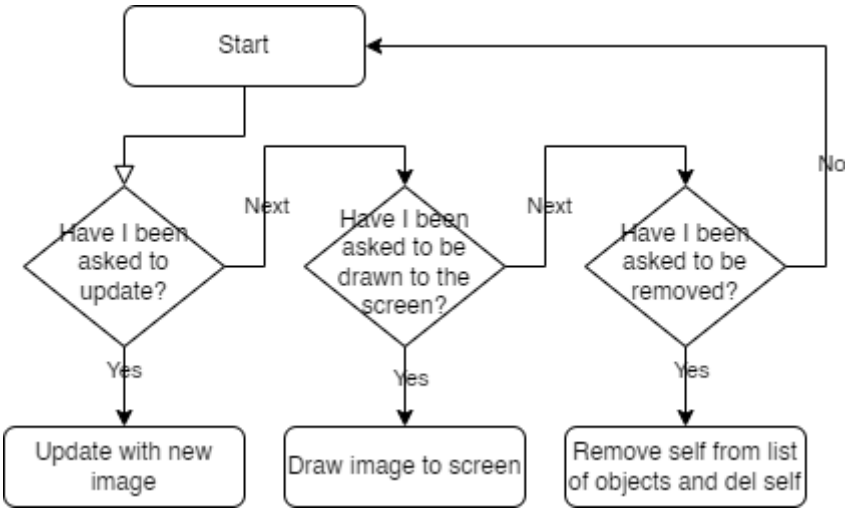
Image

Another essential necessity of GUIs is the ability to render images on the screen, if it wasn't possible then it would be significantly harder to display certain information to the user. An image class where dimensions of the image, position, as well as the name helps to make drawing images on the screen a lot easier. Making it a class helped to make all images on the screen behave in the same way as well as creating multiple instances.

The image class will have three major methods, update will allow the image class to change the image it is displaying, drawing the image it has to the screen, and lastly remove so that it can be removed if needed.

The image update will just update the graphic inside the image class so that the next time the render function is called under the image class, it will be drawing the new graphic. This function will take only a single input which is the reference to the image. The render function will just draw the current graphic in that instance of the image class on the screen when it is called. Lastly, if the removal method is called, it will remove itself from its sprite group and delete itself.

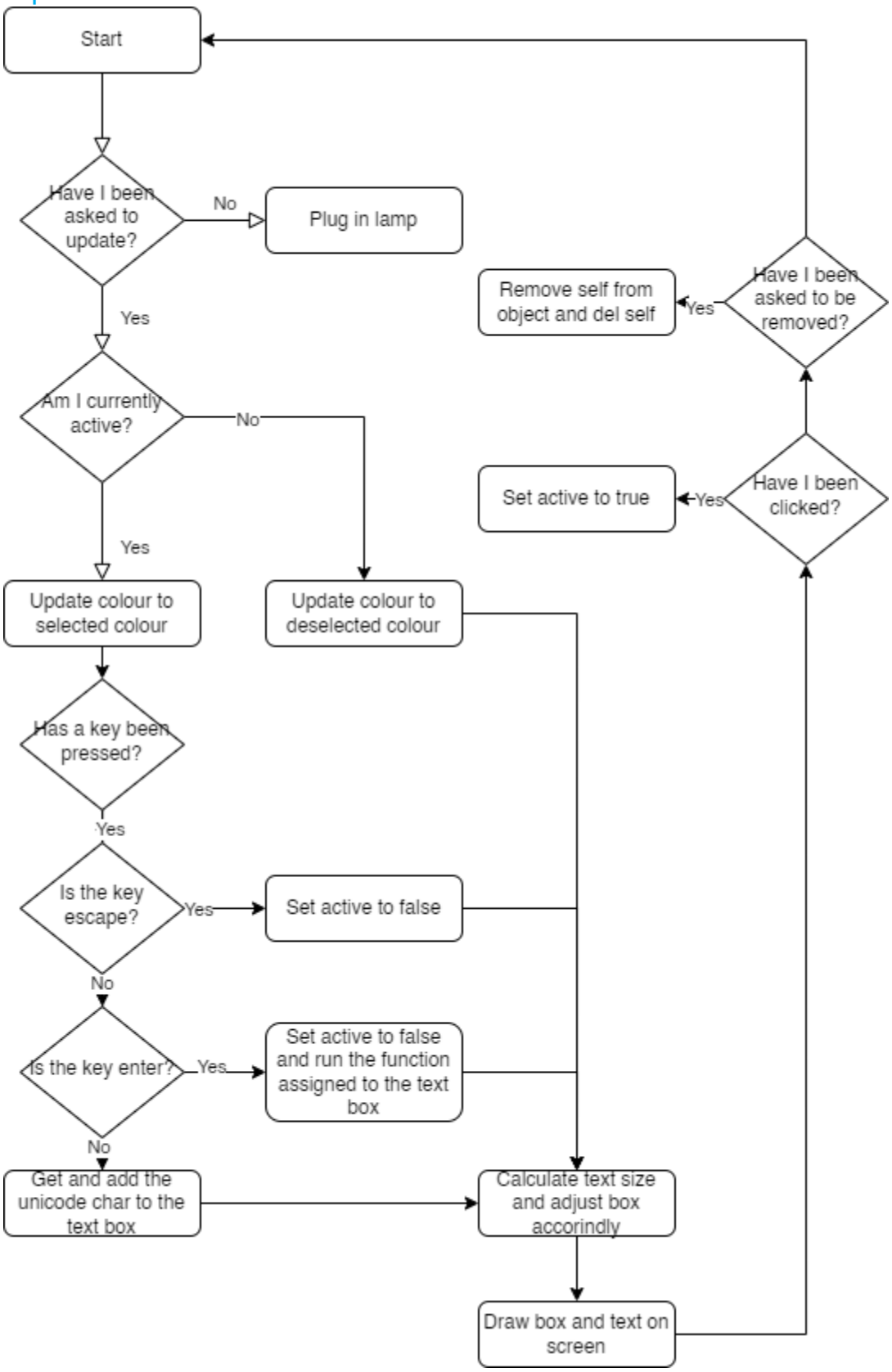
Image Flowchart



Text Input

One under looked but very needed feature is the ability to obtain user input such as text, a way to do this would be to have a text field where the user is able to write directly in to the box where the text inputted into the box would be processed. Creating a text input class will aid me in my UI design as I would be able to use it key area such as the settings menu, the join server menu where users would need to enter server information, or the in-game chat in a multiplayer game.

Text Input Flowchart

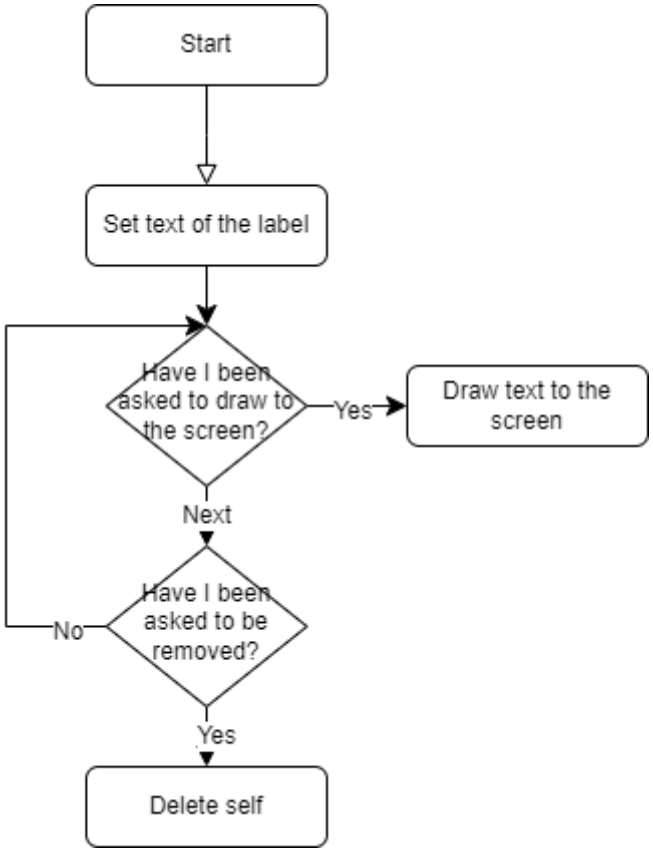


Text Label

In some parts of my GUI, I need to write text on the screen, but I don't need a surface like a button or a text label. For example, where I use these text labels is usually next to text

inputs to display and tell the user the function of the text input in the game. Most notably used in the settings menu to tell the user which box controls what value. Creating a class to handle this helps me to organise my program and make it neater, whilst also giving me the ability to create new and more text labels very easily.

Text Label Flowchart



Main

Purpose and Function

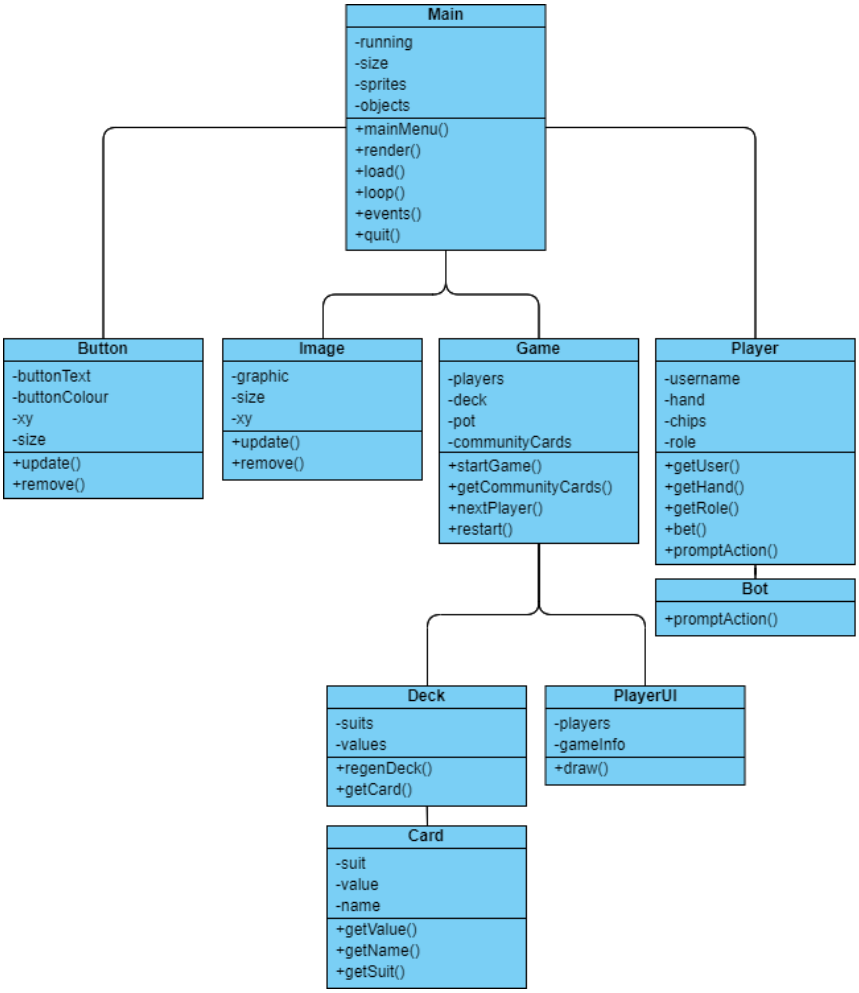
The purpose of the main file is to act as the root of the entire program, where crucial processes would be running such as rendering, loading, event checking, and other core functions.

As well as handling core functions, the main class also organises and allows other parts of the program to communicate with each other.

Initial Class Diagram

In order that I know how I should go about making my program, I will make an initial class diagram in which I can follow which will support my overall design process by having an objective to head towards.

When the program is finalised however, the class diagram might have changed, but despite that they should be similar and function in the same way.



Compare Module

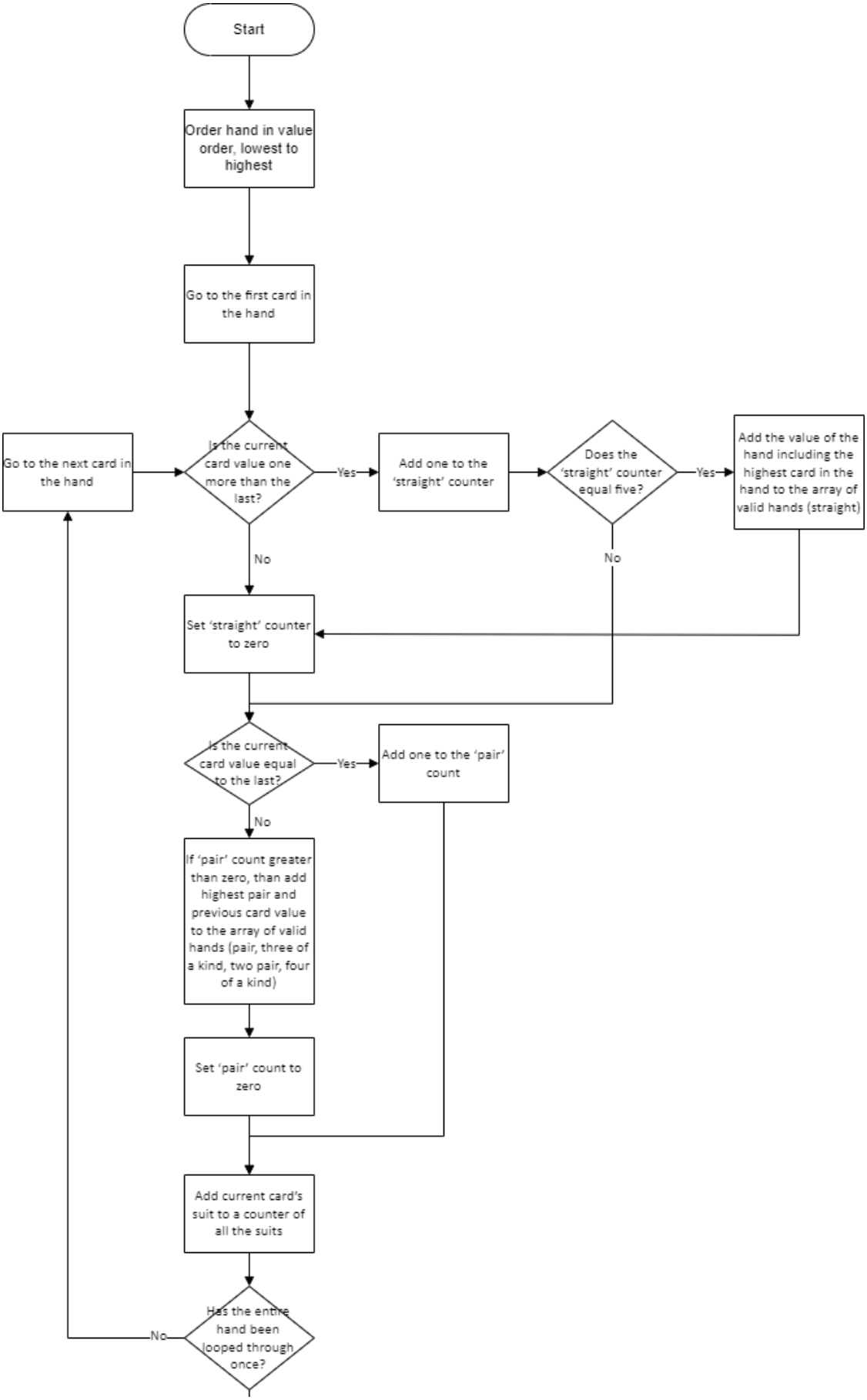
Purpose and Function

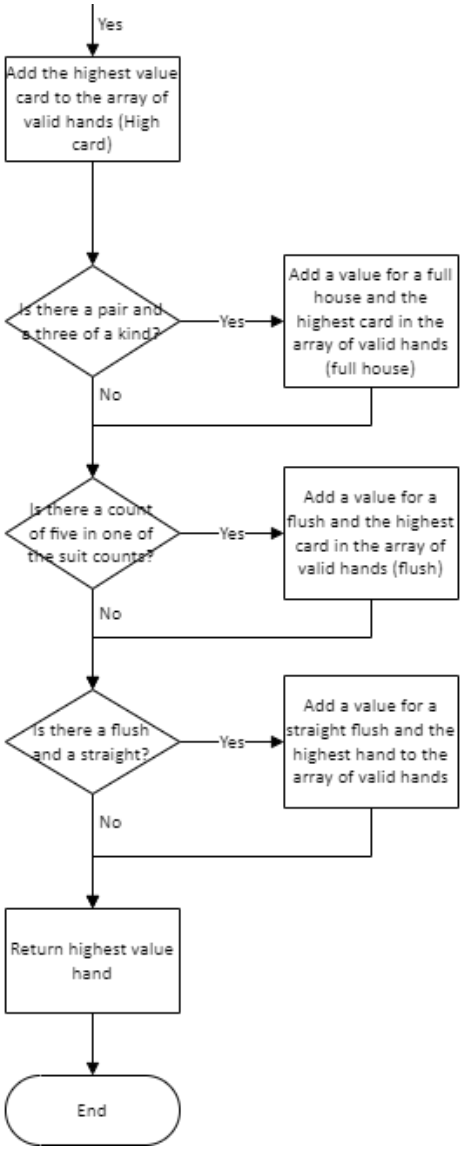
By far one of the most complex but needed parts of my program is the comparison module. This is needed so that I can easily see which player in a game has the better hand out of all the other players which is critical in a poker game as it is needed to see who wins the money in the pot. To do this, I require that I convert a player's hand into a numerical value in which I can compare against using normal operators such as less than or greater than. However, this is more easily said than done. Additionally, my personal requirements on keeping the check not too taxing on the program by ideally reducing the time complexity of the function to O(n). It would be best to compare and get every hand possible and return the best hand in one loop.

The function only takes in a single input, being an array containing the hand which contains card objects, then returns a numerical value. This can be converted into a hand name if needed using a function with a table and formatting to get the hand name back. The function returning a value is essential to me as it allows me to compare other players hands easily using normal operators such as less than and greater than, or even functions such as max().

To help aid my design of my function and additionally make it more understandable for other people looking at my program. I will make an initial flowchart outlining the key functions and methods to get the function to the correctly operate.

Flowchart


Start
Order hand in value order, lowest to highest
Go to the first card in the hand
Go to the next card in the hand
Is the current card value one more than the last?
Yes
Add one to the 'straight' counter
Does the 'straight' counter equal five?
Yes
Add the value of the hand including the highest card in the hand to the array of valid hands (straight)
No
No
Set 'straight' counter to zero
Is the current card value equal to the last?
Yes
Add one to the 'pair' count
No
If 'pair' count greater than zero, than add highest pair and previous card value to the array of valid hands (pair, three of a kind, two pair, four of a kind)
Set 'pair' count to zero
Add current card's suit to a counter of all the suits
Has the entire hand been looped through once?
No



Asset loader Module

Purpose and function

For my project to use images or any assets, it is necessary that I specify the file path where the asset I am using is located. Writing ‘asset.png’ when wanting to specify that image would be ideal, but the program wouldn’t know where to locate the image as the file path is not specified.

However, this can be easily solved if I create a function which creates a dictionary where I could reference the file name and it were to return the file path for that image.

This would be done by going through a dictionary of arrays where each array is a folder, and in each array is the images in that folder. Then using this I can easily add new folders or new images to folders. Then a function called load would create a dictionary with images where the index is the file name, and the item is the file path. This would be done by going through the first dictionary and indexing the file path with the file name.

Client Module (networking)

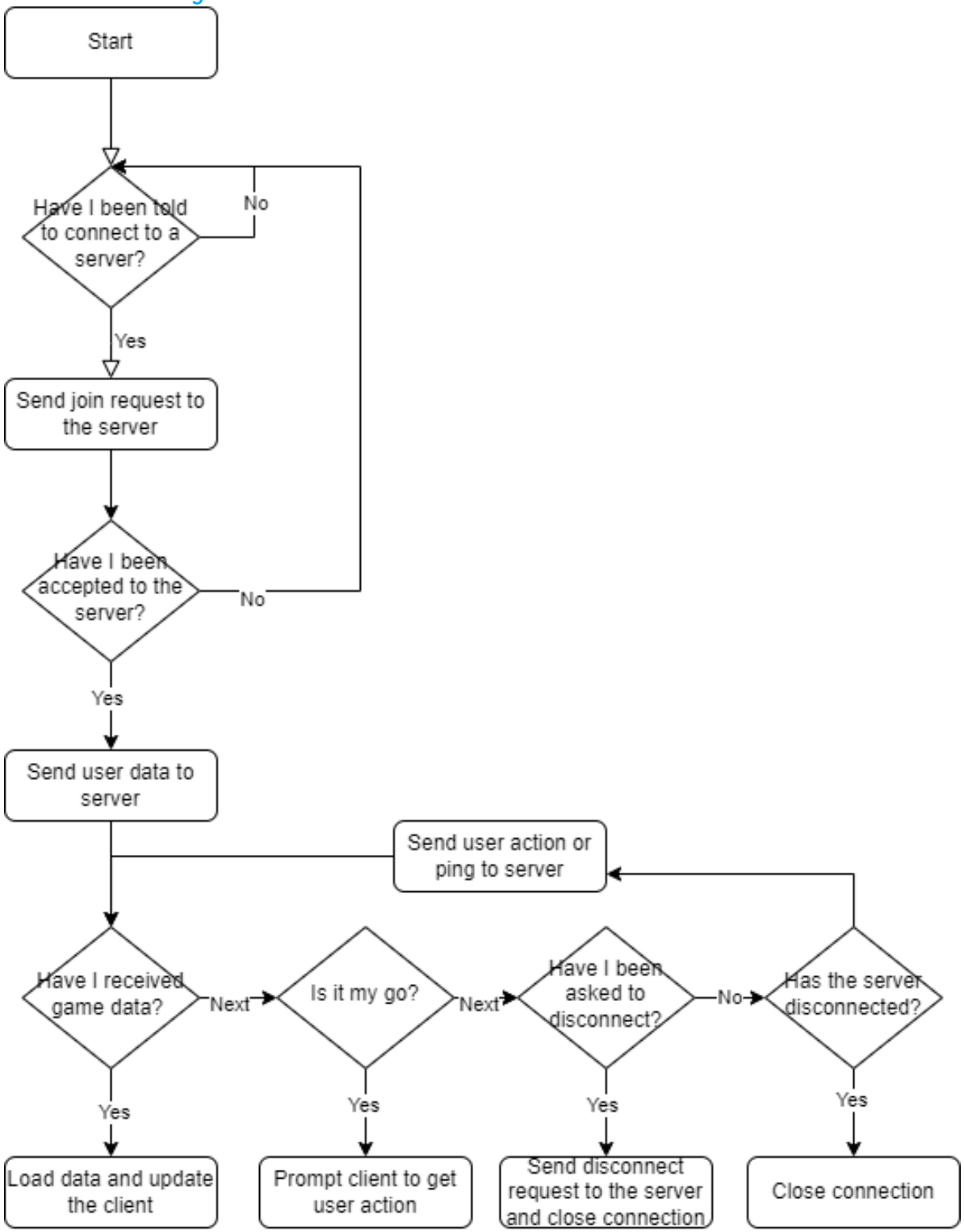
Client Networking

In order to play multiplayer, it is essential one part of my program is dedicated to client networking and server networking.

The client network needs to communicate and receive data from the server. This includes information such as current players in the game, their money, the community cards etc. In order for that to work, I need to make so that the client communicates to the server and then receives data about the game. Doing this ensures that the server remains reliable, doing it this way is good as it is the main way servers and clients communicate with each other.

The client networking when it receives data from the server such as player list or hand, needs to update the information on the client so that it can be drawn on the screen and the player can react to the game on screen. The client networking also needs to send the user actions to the server so that the server can process the request and update all the other clients.

Client Networking Flowchart



Server Module (networking)

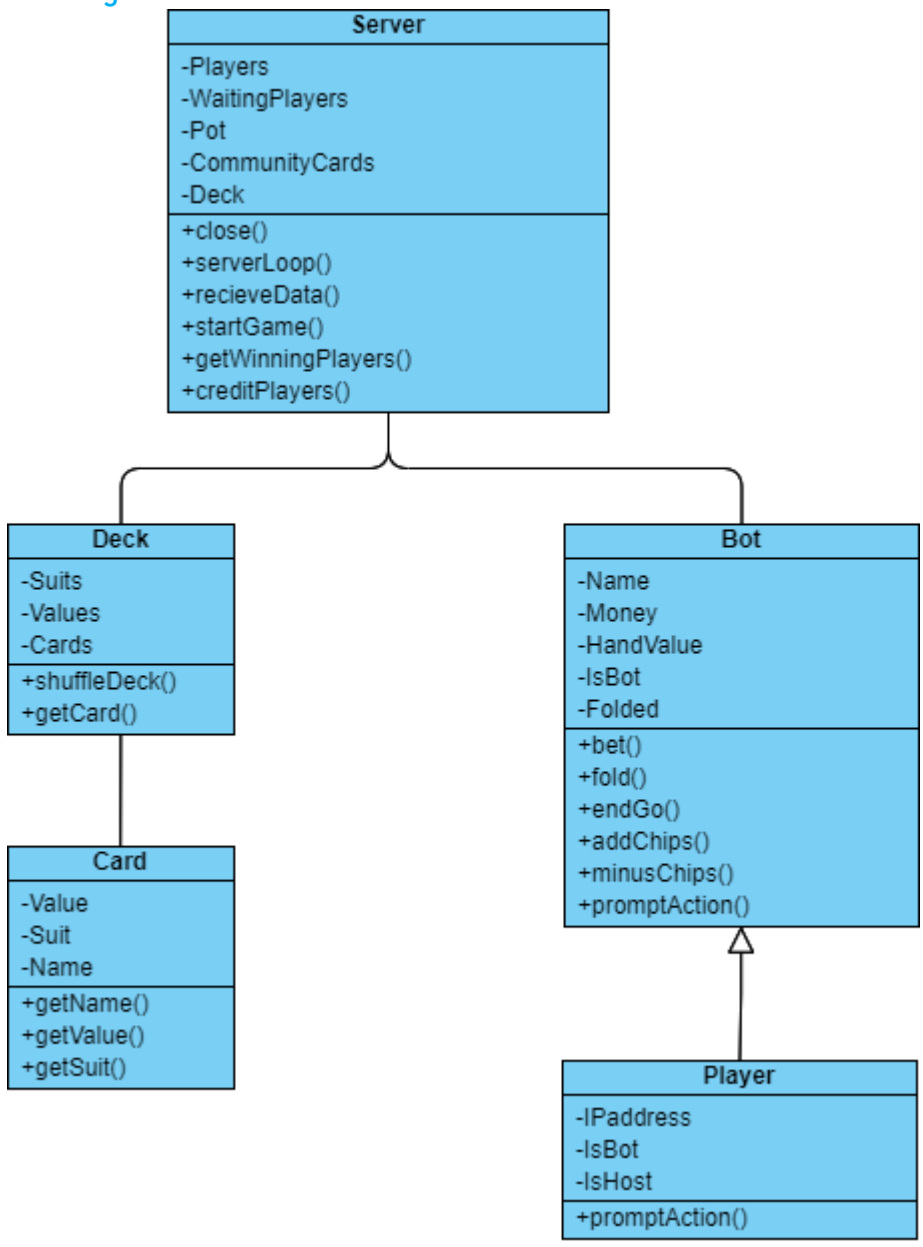
Server Networking

The server networking is probably by far one of the most complicated parts of the program, the server needs to process the game and the players whilst also receiving data and sending data from different clients. This is easily done by creating a thread for each client connected to the server waiting for data to be received from the client.

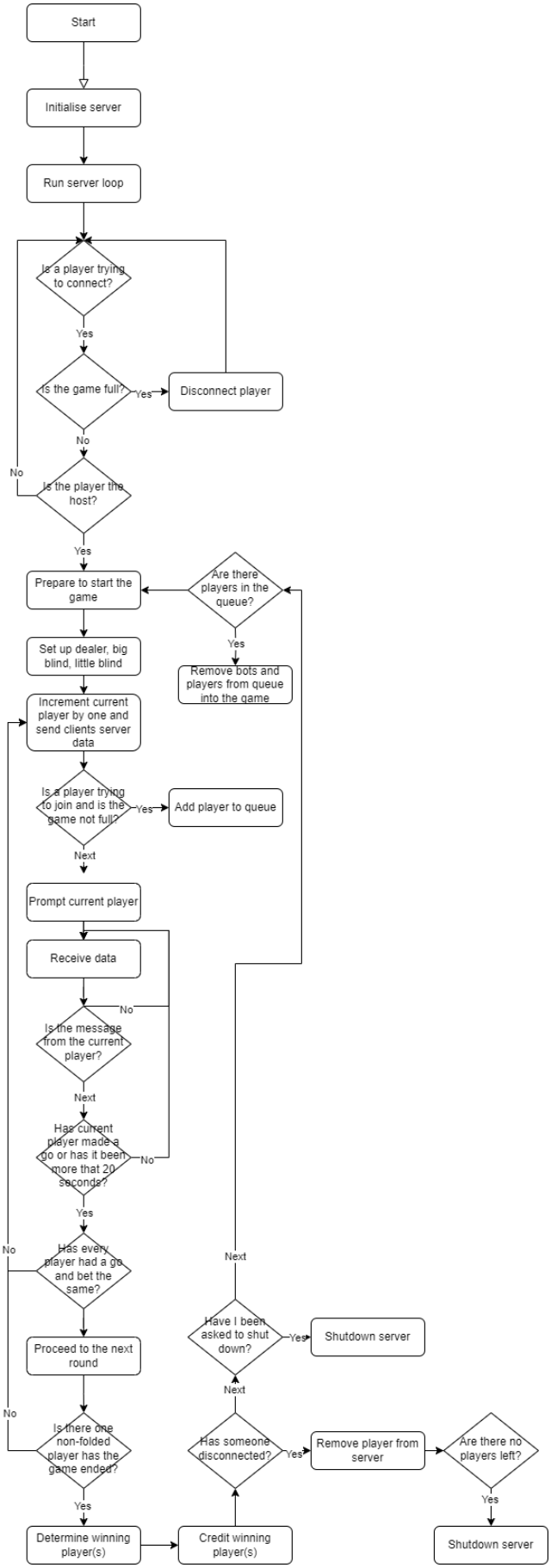
The server will need to keep track of the players connected and player waiting to play. In addition to connections and disconnections. The player data as well as the player bots would most likely be stored as classes on the server. The player class would probably inherit from the bot class or the other way around. This is because both classes would be very similar having the same values such as 'hand' and 'money '. However, the player class would be slightly different as it would have to contain the Ip address of the client and networking functions under the class such as sending a packet to the player to tell the client that they can play.

In addition to having to manage the clients, the server also needs to manage the game logic of the server involving, figuring out the current player, recording current betting amount, allowing the current user to play, drawing players hands and the community cards, and figuring out the winner of the game and rewarding the winning player(s).

Server Class Diagram



Server Networking Flowchart



Bots

The bots in the game use various amounts of data to decide whether they would like to fold, call, or raise the bet. They take into consideration their starting hand odds shown in the table below, their money, and their overall confidence to help inform these decisions.

	A	K	Q	J	T	9	8	7	6	5	4	3	2
A	85%	68%	67%	66%	66%	64%	63%	63%	62%	62%	61%	60%	59%
K	66%	83%	64%	64%	63%	61%	60%	59%	58%	58%	57%	56%	55%
Q	65%	62%	80%	61%	61%	59%	58%	56%	55%	55%	54%	53%	52%
J	65%	62%	59%	78%	59%	57%	56%	54%	53%	52%	51%	50%	50%
T	64%	61%	59%	57%	75%	56%	54%	53%	51%	49%	49%	48%	47%
9	62%	59%	57%	55%	53%	72%	53%	51%	50%	48%	46%	46%	45%
8	61%	58%	55%	53%	52%	50%	69%	50%	49%	47%	45%	43%	43%
7	60%	57%	54%	52%	50%	48%	47%	67%	48%	46%	45%	43%	41%
6	59%	56%	53%	50%	48%	47%	46%	45%	64%	46%	44%	42%	40%
5	60%	55%	52%	49%	47%	45%	44%	43%	43%	61%	44%	43%	41%
4	59%	54%	51%	48%	46%	43%	42%	41%	41%	41%	58%	42%	40%
3	58%	54%	50%	66%	45%	43%	40%	39%	39%	39%	38%	55%	39%
2	57%	53%	49%	47%	44%	42%	40%	37%	37%	37%	36%	35%	51%

Technical Solution

Key Algorithms and Classes – main.py

Overview

The main.py file contains a majority of the program and is where the game handles rendering information to the screen as well as storing and retrieving data.

Main Class

The main class controls the rendering and other vital functions such as loading the settings, changing the screen such as the menu being shown. The class also contains the main game loop where the main game logic is handled. The class when it is created, sets variables relating to the game up such as the size of the screen and the name of the game, and most importantly sets the games running status to true. This allows the main game loop to be running allow the game to render and process things. If the variable is set to false, the loop will stop and the game will close. The main class also handles the users client information and menus such as the main menu and settings.

Main Class Code:

```
class Main():
    images = []

    def __init__(self):
        pygame.init()
        self.clock = pygame.time.Clock()
        self.info = pygame.display.Info()
        self.width, self.height = self.info.current_w, self.info.current_h
        self.size = [self.width, self.height]
        self.images = {}
        self.running = True
        self.sprites = pygame.sprite.Group()
        self.objects = []
        self.game = None

        self.Card = Card


        if settings["Fullscreen"]:
            self.window = pygame.display.set_mode(self.size, pygame.NOFRAME
| pygame.HWSURFACE | pygame.DOUBLEBUF)
        else:
            self.window = pygame.display.set_mode((int(settings['Width']),
int(settings['Height'])), pygame.RESIZABLE)
            self.size = [settings['Width'], settings['Height']]

        self.grey = (20, 20, 20)
        self.green = (15, 138, 19)

        self.load()
        self.client = client.Client(username=settings['Username'],
pfp=settings['Picture'], main=self)



        pygame.display.set_caption("Pixel Poker")
        pygame.display.set_icon(self.images['Poker icon'])

        self.mainMenu()
```

```
        self.loop()

    def updateSettings(self):
        with open('settings.txt', 'w') as f:
            f.write(json.dumps(settings))

    def writeSettings(self, param, data):
        settings[param] = data
        if param == 'Username':
            self.client.username = data
        elif param == 'Picture':
            self.client.pfp = data
        self.updateSettings()

    def updateFullscreen(self):
        if settings['Fullscreen']:
            self.writeSettings('Fullscreen', False)
        else:
            self.writeSettings('Fullscreen', True)

    def events(self):
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                self.running = False
            elif event.type == pygame.KEYDOWN:
                if event.key == pygame.K_ESCAPE:
                    self.quit()


                if self.game:
                    if self.client.turn:
                        if event.key == pygame.K_RETURN:
                            self.client.bet(self.client.betting)

            pos = pygame.mouse.get_pos()
            for Object in self.objects:
                if type(Object) == TextInput:
                    if event.type == pygame.MOUSEBUTTONDOWN:
                        if Object.rect.collidepoint(pos):
                            Object.active = True
                        else:
                            Object.active = False
                            Object.call_back()
                    elif event.type == pygame.KEYDOWN:
                        if event.key == pygame.K_BACKSPACE and
Object.active:

                            Object.textInput = Object.textInput[:-1]
                        elif event.key == pygame.K_RETURN and
Object.active:
                            Object.active = False
                            Object.call_back()
                        elif event.key == pygame.K_ESCAPE and
Object.active:
                            Object.active = False
                        elif Object.active and len(Object.textInput) <=
Object.charLimit:
                            Object.textInput += event.unicode

                elif type(Object) == Button:
```

```
                    if event.type == pygame.MOUSEBUTTONDOWN:
                        if event.button == 1:
                            buttonPressed = False
                            if Object.rect.collidepoint(pos) and not
buttonPressed:
                                Object.call_back()
                                buttonPressed = True

    def quit(self):
        self.running = False
        self.client.disconnect()

    def render(self):
        self.window.fill(self.background)
        self.sprites.draw(self.window)

    def removeAllSprites(self):
        self.sprites.empty()
        for object in self.objects:
            object.remove()
        self.objects = []

    def load(self):
        self.imagePaths = {}
        for (name, group) in imgGroups.items():
            for item in group:
                ext = ".png"
                splitted = item.split(".")
                if len(splitted) > 1:
                    item = splitted[0]
                    ext = "." + splitted[1]

                self.images[item] = pygame.image.load(os.path.join(name,
item + ext)).convert_alpha()
                self.imagePaths[item] = os.path.join(name, item + ext)


    def mainMenu(self):
        self.removeAllSprites()
        self.background = self.grey
        self.logo = Image(self, xy=[self.width/2 - 84, 30], dim=[169,95],
group=self.sprites, image=self.images['Logo'])
        self.b1 = Button(self, xy=[self.width/2, 150],
function=self.modeSelect, text='Play', dim=[200, 50], group=self.objects)
        self.b2 = Button(self, xy=[self.width/2, 200],
function=self.settingsMenu, text='Settings', dim=[200, 50],
group=self.objects)
        self.b3 = Button(self, xy=[self.width/2, 250], function=self.quit,
text='Quit', dim=[200, 50], group=self.objects)
        self.qr = Image(self, xy=[self.width/2, 350], dim=[58,58],
group=self.sprites, image=self.images['QR'])

    def settingsMenu(self):
        self.removeAllSprites()
        self.background = self.grey
        self.userText = TextLabel(self, text='Username:', xy=[self.width/2
- 180, 150], group=self.objects)
        self.pictureText = TextLabel(self, text='Picture:',
xy=[self.width/2 - 160, 186], group=self.objects)
        self.portText = TextLabel(self, text='Port:', xy=[self.width/2 -
140, 222], group=self.objects)
```

```
        self.widthText = TextLabel(self, text='Width:', xy=[self.width/2 -
150, 318], group=self.objects)
        self.heightText = TextLabel(self, text='Height:', xy=[self.width/2
- 150, 354], group=self.objects)
        self.t1 = TextInput(self, default=settings['Username'],
xy=[self.width/2 - 100, 150], function=self.writeSettings,
param='Username', group=self.objects)
        self.t2 = TextInput(self, default=settings['Picture'],
xy=[self.width/2 - 100, 186], function=self.writeSettings, param='Picture',
group=self.objects)
        self.t3 = TextInput(self, default=settings['Server Port'],
xy=[self.width/2 - 100, 222], function=self.writeSettings, param='Server
Port', group=self.objects)
        self.b1 = Button(self, xy=[self.width/2, 287],
function=self.updateFullscreen, text='Fullscreen Toggle' , dim=[250, 50],
group=self.objects)
        self.fScreentxt = TextLabel(self, text='(restart to take effect)',
xy=[self.width/2 + 290, 272], group=self.objects)
        self.t4 = TextInput(self, default=settings['Width'],
xy=[self.width/2 - 100, 318], function=self.writeSettings, param='Width',
group=self.objects)
        self.t5 = TextInput(self, default=settings['Height'],
xy=[self.width/2 - 100, 354], function=self.writeSettings, param='Height',
group=self.objects)
        self.b2 = Button(self, xy=[self.width/2, 419],
function=self.mainMenu, text='Back', dim=[200, 50], group=self.objects)

    def modeSelect(self):
        self.removeAllSprites()
        self.background = self.grey
        self.logo = Image(self, xy=[self.width/2 - 84, 30], dim=[169,95],
group=self.sprites, image=self.images['Logo'])
        self.b1 = Button(self, xy=[self.width/2, 150],
function=self.singleplayer, text='Singleplayer', dim=[200, 50],
group=self.objects)
        self.b2 = Button(self, xy=[self.width/2, 200],
function=self.joinOrCreate, text='Multiplayer', dim=[200, 50],
group=self.objects)
        self.b3 = Button(self, xy=[self.width/2, 250],
function=self.mainMenu, text='Back', dim=[200, 50], group=self.objects)
        self.qr = Image(self, xy=[self.width/2, 350], dim=[58,58],
group=self.sprites, image=self.images['QR'])


    def joinOrCreate(self):
        self.removeAllSprites()
        self.background = self.grey
        self.logo = Image(self, xy=[self.width/2 - 84, 30], dim=[169,95],
group=self.sprites, image=self.images['Logo'])
        self.b1 = Button(self, xy=[self.width/2, 150],
function=self.joinServerMenu, text='Join a lobby', dim=[200, 50],
group=self.objects)
        self.b2 = Button(self, xy=[self.width/2, 200],
function=self.createMultiplayerServer, text='Create a lobby', dim=[200,
50], group=self.objects)
        self.b3 = Button(self, xy=[self.width/2, 250],
function=self.modeSelect, text='Back', dim=[200, 50], group=self.objects)
        self.qr = Image(self, xy=[self.width/2, 350], dim=[58,58],
group=self.sprites, image=self.images['QR'])

    def joinServerMenu(self):
```

```
        self.removeAllSprites()
        self.background = self.grey
        self.userText = TextLabel(self, text='Server Ip:', xy=[self.width/2
- 185, 150], group=self.objects)
        self.portText = TextLabel(self, text='Port:', xy=[self.width/2 -
150, 186], group=self.objects)
        self.t1 = TextInput(self, default='', xy=[self.width/2 - 100, 150],
group=self.objects)
        self.t2 = TextInput(self, default='', xy=[self.width/2 - 100, 186],
group=self.objects)
        self.b1 = Button(self, xy=[self.width/2, 250],
function=self.connectToServer, text='Join', dim=[200, 50],
group=self.objects)
        self.b2 = Button(self, xy=[self.width/2, 300],
function=self.joinOrCreate, text='Back', dim=[200, 50], group=self.objects)


    def connectToServer(self):
        self.background = self.green
        self.connectionIp = self.t1.textInput
        self.connectionPort = self.t2.textInput
        self.removeAllSprites()
        self.game = Game(self, gametype="Multiplayer - J")

    def createMultiplayerServer(self):
        self.removeAllSprites()
        self.background = self.green
        self.game = Game(self, gametype="Multiplayer - C")

    def singleplayer(self):
        self.removeAllSprites()
        self.background = self.green
        self.game = Game(self, gametype="Singleplayer")


    def updateObjects(self):
        for Object in self.objects:
            Object.update()


    def loop(self):
        while self.running:
            self.dt = self.clock.tick(100) / 1000
            self.events()
            self.render()
            self.updateObjects()
            pygame.display.flip()
        pygame.quit()
        exit()
```

Game Class

The game class handles games played by the player on the client and allows information about the game to be stored and accessed by the client. Having it done this way allows the drawing and rendering functions on the client be easily be able to access the games variables such as chips in the pot, player in the game, community cards, your hand and chips etc. The values in the class are set by the client part of the network when it receives information about the game and therefore should be drawing to the screen relatively fast.

Game Class Code

```
class Game():
    def __init__(self, parent, gametype=None, bots=None):
        self.parent = parent
        self.gametype = gametype
        self.players = []
        self.font = pygame.font.SysFont('Calibri',35)
        self.textColour = [0, 0, 0]
        self.gameStart = None

        self.pot = [0, 0, 0, 0, 0]
        self.communityCards = []
        self.lastbet = 0
        self.gameNumber = 0
        self.playerNumber = 0


        self.playerNumber = 0

        if self.gametype == "Singleplayer":
            self.server = server.Server(isInternal=True, game=self,
port=settings['Server Port'])
            connected = self.parent.client.connect('127.0.0.1',
settings['Server Port'])
        elif self.gametype == "Multiplayer - C":
            self.server = server.Server(game=self, port=settings['Server
Port'])
            connected = self.parent.client.connect('127.0.0.1',
settings['Server Port'])
        elif self.gametype == "Multiplayer - J":
            connected =
self.parent.client.connect(self.parent.connectionIp,
self.parent.connectionPort)
            self.server = False
        else:
            print('Attempted to create a gametype which isn\'t recognised')

        if not connected:
            print('Failed to connect.')
            return self.parent.mainMenu()


        self.mainPlayerHUD = MainPlayerUI(self, size=100,
picture=self.parent.client.pfp, username=self.parent.client.username)


    def setLastBet(self, amount):
        self.lastbet = amount

    def getLastBet(self):
        return self.lastbet
```

```
    def getPlayers(self):
        return self.players

    def getCommunityCards(self):
        return self.communityCards
```

Card Class

The card class handles drawing and displaying cards on the screen, as well as handling during hand comparisons. The card has a suit and value. The suits of the cards are defined by a deck on the server, whereas the values and names for the card are defined in the constructor. The card can be flipped or be blank, which is crucial for cards to be drawn on the screen.

Card Class Code

```
class Card():
    def __init__(self, parent, suit, value, xy=[100, 100], size=[100,
100]):
        self.parent = parent
        self.suit = suit
        self.value = value
        self.xy = xy
        self.size = size
        self.values = {
            1 : "2",
            2 : "3",
            3 : "4",
            4 : "5",
            5 : "6",
            6 : "7",
            7 : "8",
            8 : "9",
            9 : "10",
            10 : "Jack",
            11 : "Queen",
            12 : "King",
            13 : "Ace"
        }
        if self.value:
            self.name = self.values[self.value] + " of " + self.suit
            self.card = self.parent.images[self.name]
        else:
            self.card = self.parent.images['card_empty']
            self.name = None
        self.back = self.parent.images['card_back']

        self.cardShown = False


    def switch(self):
        if self.value:
            if self.image.graphic == self.card:
                self.image.update(self.back)
            else:
                self.image.update(self.card)
        else:
            if self.image.graphic == self.card:
                self.image.update(self.back)
            else:
                self.image.update(self.parent.images['card_empty'])
```

```
    def getValue(self):
        return self.value

    def setValue(self, nValue):
        self.value = nValue

    def getName(self):
        return self.name

    def getValueName(self):
        return self.values[self.value]

    def setName(self):
        self.name = self.values[self.value] + " of " + self.suit

    def getSuit(self):
        return self.suit

    def setSuit(self, nSuit):
        self.suit = nSuit

    def getXY(self):
        return self.xy

    def setXY(self, nXY):
        self.xy = nXY

    def show(self):
        self.image = Image(self.parent, xy=self.xy, dim=self.size,
group=self.parent.sprites, image=self.card)
        self.cardShown = True

    def hide(self):
        self.image.kill()
        self.cardShown = False

    def remove(self):
        self.image.kill()
        del self
```

Button Class

The button is a crucial part of any GUI therefore I had to include it. The class is created with few parameters by default that can be changed, for example the text in the button, the font, size, position, and colour. The button can also be supplied a function to be ran when it is clicked. This is crucial as otherwise the button would be useless.

Button Class Code

```
class Button():
    def __init__(self, parent, text='Button', font='Calibri', dim=[100,50],
xy=[0,0], textColour=[0, 0, 0], buttonColour=[122, 122, 122],
function=None, args=None, group=None):
        self.buttonText = text
        self.font = pygame.font.SysFont(font,35)
        self.buttonDim = dim
        self.xy = xy
        self.textColour = textColour
        self.buttonColour = buttonColour
        self.buttonActive = self.getButtonColours()
        self.function = function
        self.pressed = False
        self.parent = parent
        self.group = group
        self.args = args

        self.image = pygame.Surface((self.buttonDim[0], self.buttonDim[1]))
        self.rect = self.image.get_rect(center=(self.xy[0], self.xy[1]))

        self.text = self.font.render(self.buttonText, True,
self.textColour)

        self.group.append(self)

    def colourChange(self):
        self.buttonColour = self.buttonActive[0]
        pos = pygame.mouse.get_pos()
        if self.rect.collidepoint(pos):
            self.buttonColour = self.buttonActive[1]

    def update(self):
        self.colourChange()
        mouse = pygame.mouse.get_pos()
        self.image.fill((self.pressed and self.buttonActive[0]) or
self.buttonColour)
        self.image.blit(self.text, [self.rect.width/2 -
self.text.get_rect().width/2, self.rect.height/2 -
self.text.get_rect().height/2])
        self.parent.window.blit(self.image, self.rect)

    def getButtonColours(self):
        original = self.buttonColour
        highlight = []
        click = []
        for value in self.buttonColour:
            highlight.append(value + 50)
            click.append(value - 50)
        return [original, highlight, click]

    def call_back(self):
        self.buttonColour = self.buttonActive[2]
        if self.function:
```

```
            if self.args:
                self.function(self.args)
            else:
                self.function()

    def remove(self):
        global buttonList
        if self in self.group:
            self.group.remove(self)
        else:
            print('Im already gone')
```

Text Input Class

The text input is another key part of GUI, especially when it comes to the settings. It allows the player to write inside of the box where the text in the box can be used to set and process things. When a text input is created it takes certain parameters such as the font, colours, the starting text in the input, position, and size. The input box when click on enables the boxes active status, allowing the user to type in the box, if the user then clicks off the input box or presses escape, then the active status is set to false and the user is no longer able to type any longer. If the button is connected do a function, upon pressing enter the text in the box is parsed into the link function along with any other addition parameters defined in the button.

Text Input Class Code

```
class TextInput():
    def __init__(self, parent, font='Calibri', currentColour=(50, 50, 50),
selectColour=(100, 100, 100), default='', xy=[100, 100], size=[140, 35],
function=None, param=None, group=None):
        self.size = size
        self.xy = xy
        self.font = pygame.font.SysFont(font, self.size[1])
        self.textInput = str(default)
        self.rect = pygame.Rect(xy[0], xy[1], self.size[0], self.size[1])
        self.selectedColour = selectColour
        self.passiveColour = currentColour
        self.currentColour = self.passiveColour
        self.charLimit = 30
        self.active = False
        self.group = group
        self.parent = parent
        self.group.append(self)
        self.function = function
        self.param = param

    def update(self):
        if self.active:
            self.currentColour = self.selectedColour
        else:
            self.currentColour = self.passiveColour

        pygame.draw.rect(self.parent.window, self.currentColour, self.rect)
        inputtedText = self.font.render(self.textInput, True, (255, 255,
255))
        self.rect.w = max(100, inputtedText.get_width()+10)
        self.parent.window.blit(inputtedText, (self.rect.x + 5, self.rect.y
+ 5))

    def call_back(self):
        if self.function and self.param:
```

```
            self.function(self.param, self.textInput)

    def remove(self):
        if self in self.group:
            self.group.remove(self)
        else:
            print('Im already gone')
```

Image Class

The image class is another key part of the GUI, it is used in a lot of places for example in GUIs and card rendering on the screen in gameplay. It takes basic parameters being the specified image, the position and the size. If the image is unable to be loaded for a particular reason, a 'no image' image will take its place instead. The image also has the ability to be updated with a new image, allowing for an image on the screen to change appearance, this is especially useful when the players cards are able to be flipped.

Image Class Code

```
class Image(pygame.sprite.Sprite):
    def __init__(self, parent, image=None, xy=[0,0], dim=[100,100],
group=None):
        super().__init__()
        self.parent = parent
        self.graphic = image
        self.xy = xy
        if self.graphic != None:
            self.image = pygame.transform.scale(self.graphic, (dim[0],
dim[1]))
        else:
            self.graphic = self.parent.images['no image']
            self.image = pygame.transform.scale(self.graphic, (dim[0],
dim[1]))
        self.rect = self.image.get_rect(topleft=(self.xy[0], self.xy[1]))
        self.dim = dim
        self.group = parent.sprites


        self.group.add(self)

    def update(self, image):
        self.graphic = image
        if self.graphic != None:
            self.image = pygame.transform.scale(self.graphic, (self.dim[0],
self.dim[1]))
            self.rect = self.image.get_rect(topleft=(self.xy[0],
self.xy[1]))

    def remove2(self):
        if self in self.group:
            self.group.remove(self)
            self.kill()
        else:
            print('Im already gone')
```

Text Label Class

The text label class is a needed piece of GUI showing the player text on the screen in situations where some type of other GUI class is unable to showcase the text. Used primarily in the main menus, it takes a few inputs being colour, size, position, font, and text.

Text Label Class Code

```
class TextLabel():
    def __init__(self, parent, text='', font='Calibri', xy=[0,0],
textColour=(255, 255, 255), size=35, group=None):
        self.size = size
        self.font = pygame.font.SysFont(font, self.size)
        self.parent = parent
        self.text = text
        self.xy = xy
        self.group = group
        self.colour = textColour

        self.renderText = self.font.render(self.text, True, textColour)
        self.renderRect = self.renderText.get_rect()

        self.group.append(self)

    def update(self):
        self.parent.window.blit(self.renderText, (self.xy[0] -
self.renderRect.w/2, self.xy[1]), )

    def remove(self):
        if self in self.group:
            self.group.remove(self)
        else:
            print('Im already gone')
```

Player Info Class

The player info class is a part of the player GUI and draws the player information on the screen for the player to see. The class takes in three main parameters, that being, the player it is going to draw the information for, the position of the displayed information, and the size. The class has many different drawing functions relating to drawing the player information on screen, such as drawing the chips, their role, and their name and picture.

Player Info Class Code

```
class PlayerInfo():
    def __init__(self, parent, player=None, xy=[100, 100], font='Calibri',
textColour=[0,0,0], size=100):
        self.parent = parent
        self.player = player
        self.xy = xy
        print(self.player)
        self.money = self.player['Money']
        self.username = self.player['Name']
        self.graphic = None
        self.group = self.parent.objects
        self.size = size
        self.chips = []
        self.roleIcon = None
        self.action = 'jarong'


        self.textColour = textColour
        self.font = pygame.font.SysFont(font, int(self.size * 0.35))
```

```
        self.text = self.font.render(self.username, True, self.textColour)

        self.profilePicture = Image(self.parent, image=self.graphic,
xy=self.xy, dim=[self.size, self.size])

        self.group.append(self)
        self.drawChips()
        self.drawRole()

    def drawChips(self):
        value = 1
        chipSpacingX = self.size / 4
        chipSpacingY = self.size / 50
        chipXY = [self.xy[0] + (self.size * 1.1),  self.xy[1] + (self.size
/ 2)]
        if self.chips != []:
            for chip in self.chips:
                chip.kill()
        self.chips = []
        for index in self.money:
            for x in range(index):
                self.chips.append(Image(self.parent,
image=self.parent.images[str(value)], xy=(chipXY[0], chipXY[1]),
dim=[self.size / 4, self.size / 4]))
                chipXY[1] -= chipSpacingY
            chipXY[0] += chipSpacingX
            chipXY[1] = self.xy[1] + (self.size / 2)
            value *= 2

    def drawRole(self):
        iconXY = [self.xy[0] + 100,  self.xy[1] + 100]
        if self.roleIcon != None:
            self.roleIcon.kill()
        playerRole = self.player['Role']
        if not playerRole == 'Player':
            if playerRole == 'Dealer':
                self.roleIcon = Image(self.parent,
image=self.parent.images['Dealer'], xy=(iconXY[0], iconXY[1]),
dim=[self.size / 4, self.size / 4])
            elif playerRole == 'Little Blind':
                self.roleIcon = Image(self.parent,
image=self.parent.images['Little blind'], xy=(iconXY[0], iconXY[1]),
dim=[self.size / 4, self.size / 4])
            elif playerRole == 'Big Blind':
                self.roleIcon = Image(self.parent,
image=self.parent.images['Big blind'], xy=(iconXY[0], iconXY[1]),
dim=[self.size / 4, self.size / 4])


    def update(self):
        self.actionText = self.font.render(self.action, True,
self.textColour)
        self.parent.window.blit(self.actionText, (self.xy[0] + (self.size *
1.1), self.xy[1] + 80), )
        if self.text:
            self.parent.window.blit(self.text, (self.xy[0] + (self.size *
1.1), self.xy[1]), )
        self.drawChips()
        self.drawRole()

    def remove2(self):
```

```
        if self in self.group:
            self.group.remove(self)
        else:
            print('Im already gone')
```

Main Player UI Class

The main player ui class handles the rendering of all the important information on the screen. This include the controls, the community cards, your hand, other players, and the game information such as time and pot.

Main Player UI Code

```
class MainPlayerUI():
    def __init__(self, parent, username='Username', picture=None,
hand=[None, None], font='Calibri', textColour=[0,0,0], size=100):
        self.parent = parent
        self.size = 100
        self.xy = [parent.parent.width/2 - self.size * 2.5, 3 *
parent.parent.height / 4]
        self.hand = []
        self.overallHand = []
        self.username = username
        try:
            self.graphic = pygame.image.load('./User Images/' +
picture).convert_alpha()
        except:
            self.graphic = self.parent.parent.images['no image']
        self.group = self.parent.parent.objects
        self.group.append(self)
        self.handShown = 0
        self.chips = []
        self.potChips = []
        self.emptyCards = []
        self.roleIcon = None
        self.role = 'Player'
        self.totalMoney = 0
        self.playerDisplays = {}

        self.textColour = textColour
        self.font = pygame.font.SysFont(font, int(self.size * 0.35))

        self.bettingAmount = 0
        self.action = "Check"
        self.foldAction = "Fold"
        self.handAction = "Check Hand"
        self.chatText = 'Toggle Chat'
        self.handName = ''
        self.timeElapsed = '0'
        self.foldActionText = self.font.render(self.foldAction, True,
self.textColour)
        self.handActionText = self.font.render(self.handAction, True,
self.textColour)
        self.pokerHelp = False
        self.pokermenu = None
        self.paused = False
        self.lastTime = 0
        self.lastTime2 = 0
        self.b1 = None
        self.textInput = None
        self.chatMessages = ['mate', 'john', 'egg']
        self.chatting = True
        self.messages = []
```

```
        self.textObjects = []

        self.drawSelfPlayerInfo()
        self.drawControls()
        self.drawHand()
        self.drawPlayers()
        self.drawCommunityCards()
        self.drawRole()

    def drawPokerHelpMenu(self):
        if self.pokerHelp:
            if not self.pokermenu:
                self.pokermenu = Image(self.parent.parent,
image=self.parent.parent.images['Poker Assistance'],
dim=[self.parent.parent.width/2, self.parent.parent.height],
xy=[self.parent.parent.width/2 - self.parent.parent.width/4, 0])
        elif self.pokermenu:
            self.pokermenu.remove2()
            self.pokermenu = None

    def drawSelfPlayerInfo(self):
        self.profilePicture = Image(self.parent.parent, image=self.graphic,
xy=self.xy, dim=[self.size, self.size])
        self.text = self.font.render(self.username, True, self.textColour)


    def drawChips(self):
        value = 1
        chipSpacingX = self.size / 1.5
        chipSpacingY = 4 * self.size / 80
        chipXY = [self.xy[0] + (self.size * 1.1),  self.xy[1] + (self.size
/ 1.5)]
        if self.chips != []:
            for chip in self.chips:
                chip.kill()
        self.chips = []
        for index in self.parent.parent.client.money:
            for x in range(index):
                self.chips.append(Image(self.parent.parent,
image=self.parent.parent.images[str(value)], xy=(chipXY[0], chipXY[1]),
dim=[self.size / 1.5, self.size / 1.5]))
                chipXY[1] -= chipSpacingY
            chipXY[0] += chipSpacingX
            chipXY[1] = self.xy[1] + (self.size / 1.5)
            value *= 2

    def calculatePot(self):
        value = 1
        totalMoney = 0
        for i in self.parent.pot:
            amount = value * i
            totalMoney += amount
            value *= 2
        return totalMoney

    def drawPot(self):
        value = 1
        chipSpacingX = self.size / 1.5
        chipSpacingY = 4 * self.size / 80
        chipY = self.parent.parent.height / 2
        chipXY = [self.parent.parent.width/2 - self.size * 2,  chipY]
```

```
        if self.potChips != []:
            for chip in self.potChips:
                chip.kill()
        self.potChips = []
        for index in self.parent.pot:
            for x in range(index):
                self.potChips.append(Image(self.parent.parent,
image=self.parent.parent.images[str(value)], xy=(chipXY[0], chipXY[1]),
dim=[self.size / 1.5, self.size / 1.5]))
                chipXY[1] -= chipSpacingY
            chipXY[0] += chipSpacingX
            chipXY[1] = chipY
            value *= 2

        self.potValue = self.font.render('Pot: £' +
str(self.calculatePot()), True, self.textColour)
        self.potVRect = self.potValue.get_rect()

    def drawRole(self):
        iconXY = [self.xy[0] + 100,  self.xy[1] + 100]
        if self.roleIcon != None:
            self.roleIcon.kill()
        playerRole = self.role
        if not playerRole == 'Player':
            if playerRole == 'Dealer':
                self.roleIcon = Image(self.parent.parent,
image=self.parent.parent.images['Dealer'], xy=(iconXY[0], iconXY[1]),
dim=[self.size / 4, self.size / 4])
            elif playerRole == 'Little Blind':
                self.roleIcon = Image(self.parent.parent,
image=self.parent.parent.images['Little blind'], xy=(iconXY[0], iconXY[1]),
dim=[self.size / 4, self.size / 4])
            elif playerRole == 'Big Blind':
                self.roleIcon = Image(self.parent.parent,
image=self.parent.parent.images['Big blind'], xy=(iconXY[0], iconXY[1]),
dim=[self.size / 4, self.size / 4])

    def drawControls(self):
        self.xpos = self.parent.parent.width
        self.ypos = self.parent.parent.height - 160
        self.upArrow = Image(self.parent.parent,
image=self.parent.parent.images['Up'],xy=[self.xpos - 40, self.ypos] ,
dim=[40, 40])
        self.downArrow = Image(self.parent.parent,
image=self.parent.parent.images['Down'], xy=[self.xpos - 80, self.ypos],
dim=[40, 40])
        self.enter = Image(self.parent.parent,
image=self.parent.parent.images['Enter'], xy=[self.xpos - 40, self.ypos +
40], dim=[40, 40])
        self.fKey = Image(self.parent.parent,
image=self.parent.parent.images['F'], xy=[self.xpos - 40, self.ypos + 80],
dim=[40, 40])
        self.spacebar = Image(self.parent.parent,
image=self.parent.parent.images['Space'], xy=[self.xpos - 40, self.ypos +
120], dim=[40, 40])
        self.tabkey = Image(self.parent.parent,
image=self.parent.parent.images['Tab'], xy=[self.xpos - 40, self.ypos -
40], dim=[40, 40])
        if self.parent.server:
            if not self.parent.server.isInternal:
```

```
                self.altkey = Image(self.parent.parent,
image=self.parent.parent.images['Alt'], xy=[self.xpos - 40, self.ypos -
80], dim=[40, 40])
        else:
            self.altkey = Image(self.parent.parent,
image=self.parent.parent.images['Alt'], xy=[self.xpos - 40, self.ypos -
80], dim=[40, 40])

    def drawHand(self):
        if len(self.hand) >= 2:
            card1 = self.hand[0]
            card2 = self.hand[1]
            card1.setXY([self.parent.parent.width/2 - 100,
self.parent.parent.height - 100])
            card2.setXY([self.parent.parent.width/2 - 30,
self.parent.parent.height - 100])
            card1.show()
            card2.show()
            card1.switch()
            card2.switch()

    def drawPlayers(self):
        players = self.parent.players
        playerIndex = 0
        for idx, val in enumerate(players):
            if val['Id'] == self.parent.parent.client.id:
                playerIndex = idx
                break
        visPlayers = players[playerIndex:] + players[:playerIndex]
        playerPositions = [
            [self.parent.parent.width/20, self.parent.parent.height/2],
            [self.parent.parent.width/20, self.parent.parent.height/12],
            [self.parent.parent.width - self.parent.parent.width/5,
self.parent.parent.height/12],
            [self.parent.parent.width - self.parent.parent.width/5,
self.parent.parent.height/2],
        ]

        self.playerDisplays = {}

        for playerNumber in range(len(visPlayers)):
            if visPlayers[playerNumber]['Id'] !=
self.parent.parent.client.id:
                playerObject = PlayerInfo(self.parent.parent,
player=visPlayers[playerNumber], xy=playerPositions[playerNumber - 1])
                self.playerDisplays[visPlayers[playerNumber]['Id']] =
playerObject


    def flipHand(self):
        self.cardXY = [self.parent.parent.width/2 - 70,
self.parent.parent.height - 100]
        if self.handShown == 0:
            self.handShown = 1
        else:
            self.handShown = 0
        for card in self.hand:
            card.switch()

    def drawCommunityCards(self):
        self.cardsxy = [self.parent.parent.width /2 - 190, 70]
```

```
        for card in self.parent.communityCards:
            card.setXY(self.cardsxy)
            card.show()
            self.cardsxy[0] += 70
        for card in range(5 - len(self.parent.communityCards)):
            blankCard = Card(self.parent.parent, None, None)
            blankCard.setXY(self.cardsxy)
            blankCard.show()
            blankCard.switch()
            self.emptyCards.append(blankCard)
            self.cardsxy[0] += 70

    def removeCommunityCards(self):
        for blank in self.emptyCards:
            blank.remove()
        self.emptyCards = []
        for card in self.parent.communityCards:
            if card.cardShown:
                card.hide()

    def gameInfo(self):
        if self.parent.parent.client.currentPlayer != None:
            self.currentPlayerText =
self.font.render(self.parent.parent.client.currentPlayer + "'s Turn", True,
self.textColour)
        else:
            self.currentPlayerText = self.font.render("Game is starting",
True, self.textColour)
        self.parent.parent.window.blit(self.currentPlayerText,
(self.parent.parent.width/2 - self.currentPlayerText.get_rect().width/2,
40))

        self.timeText = self.font.render(self.getTime() + ' Game 1, Turn
1', True, self.textColour)
        self.timeRect = self.timeText.get_rect()
        self.parent.parent.window.blit(self.timeText,
(self.parent.parent.width/2 - self.timeRect.width/2, 0))

    def getTime(self):
        if self.parent.gameStart:
            self.timeInS = math.floor(time.time() - self.parent.gameStart)
            self.minutesElapsed = math.floor(self.timeInS/60)
            self.hoursElapsed = math.floor(self.minutesElapsed/60)

            self.timer = f"{self.hoursElapsed:02}:{self.minutesElapsed -
(self.hoursElapsed * 60):02}:{self.timeInS - (self.minutesElapsed *
60):02}"
            return self.timer
        else:
            return 'Waiting For Start,'

    def togglePause(self):
        if not self.paused and time.time() - self.lastTime >= 0.1:
            self.lastTime = time.time()
            self.paused = True
        elif time.time() - self.lastTime >= 2:
            self.lastTime = time.time()
            self.b1.remove()
            self.b1 = None
            self.b2.remove()
            self.b2 = None
```

```
            self.paused = False

    def toggleChat(self):
        if not self.chatting and time.time() - self.lastTime2 >= 0.1:
            self.lastTime2 = time.time()
            self.chatting = True
        elif time.time() - self.lastTime2 >= 2:
            self.lastTime2 = time.time()
            self.textInput.remove()
            for i in self.textObjects:
                i.remove()
            self.textObjects = []
            self.textInput = None
            self.chatting = False

    def leave(self):
        self.parent.parent.client.disconnect()
        self.parent.parent.game = None
        self.parent.parent.mainMenu()

    def sendMessage(self, text):
        print('Yes you called')
        if self.textInput and text != '':
            self.parent.parent.client.networkQueue.put(b'7['+ bytes(text,
'utf-8') + b']\x1e')
            self.textInput.textInput = ''

    def chatMenu(self):
        positions = [
            [150, self.parent.parent.height - 135],
            [150, self.parent.parent.height - 170],
            [150, self.parent.parent.height - 205],
            [150, self.parent.parent.height - 240],
            [150, self.parent.parent.height - 275]
        ]
        pygame.draw.rect(self.parent.parent.window, (20, 20, 20),
pygame.Rect(50, self.parent.parent.height - 300, 270, 200))
        if not self.textInput:
            self.textInput = TextInput(self.parent.parent,
function=self.sendMessage, param='', xy=[50, self.parent.parent.height -
100], group=self.parent.parent.objects)
        for i in self.textObjects:
            i.remove()
        self.textObjects = []
        count = 0
        for message in self.chatMessages:
            self.textObjects.append(TextLabel(self.parent.parent,
text=message, xy=positions[count], group=self.parent.parent.objects))
            count += 1


    def pauseMenu(self):
        pygame.draw.rect(self.parent.parent.window, (20, 20, 20),
pygame.Rect(0, 0, self.parent.parent.width, self.parent.parent.height))
        if not self.b1 or not self.b2:
            self.b1 = Button(self.parent.parent, text='Quit',
function=self.parent.parent.quit,  xy=[self.parent.parent.width/2,
self.parent.parent.height/2],group=self.parent.parent.objects)
            self.b2 = Button(self.parent.parent, text='Main Menu',
function=self.leave, dim=[250, 50], xy=[self.parent.parent.width/2,
self.parent.parent.height/2 - 50],group=self.parent.parent.objects)
```

```
    def update(self):
        self.overallHand = self.hand + self.parent.communityCards

        self.drawPokerHelpMenu()
        self.drawPot()
        self.drawChips()
        self.gameInfo()
        self.removeCommunityCards()
        self.drawCommunityCards()
        self.drawRole()

        self.betText = self.font.render(("£" +
str(self.parent.parent.client.betting)), True, self.textColour)
        self.pokerAssistance = self.font.render("Poker Assistance", True,
self.textColour)
        self.textChat = self.font.render(self.chatText, True,
self.textColour)
        if self.parent.lastbet < self.parent.parent.client.betting:
            self.betPromptText = self.font.render("Raise", True,
self.textColour)
        elif self.parent.parent.client.betting == 0:
            self.betPromptText = self.font.render("Check", True,
self.textColour)
        elif self.parent.lastbet == self.parent.parent.client.betting:
            self.betPromptText = self.font.render("Call", True,
self.textColour)
        elif self.parent.parent.client.betting ==
self.parent.parent.client.betting:
            self.betPromptText = self.font.render("All in", True,
self.textColour)
        self.betTextRect = self.betText.get_rect()
        self.betPromptRect = self.betPromptText.get_rect()
        self.parent.parent.window.blit(self.text, (self.xy[0] + (self.size
* 1.1), self.xy[1]))
        self.parent.parent.window.blit(self.betText, (self.xpos -
self.betTextRect.width - 85, self.ypos))
        self.parent.parent.window.blit(self.pokerAssistance, (self.xpos -
280, self.ypos - 40))
        self.parent.parent.window.blit(self.betPromptText, (self.xpos -
self.betPromptRect.width - 45, self.ypos + 40))
        self.parent.parent.window.blit(self.foldActionText, (self.xpos -
105, self.ypos + 80))
        self.parent.parent.window.blit(self.handActionText, (self.xpos -
210, self.ypos + 120))
        self.parent.parent.window.blit(self.potValue,
(self.parent.parent.width - self.potVRect.width, 0))
        if self.parent.server:
            if not self.parent.server.isInternal:
                self.parent.parent.window.blit(self.textChat, (self.xpos -
280, self.ypos - 80))
        else:
            self.parent.parent.window.blit(self.textChat, (self.xpos - 280,
self.ypos - 80))
        if self.handShown == 1:
            self.handValue = compare.getValueOfHand(self.overallHand)
            self.handName = compare.valueToName(self.handValue)
            self.handNameText = self.font.render(self.handName, True,
self.textColour)
            self.handNameTextRect = self.handNameText.get_rect()
```

```
            self.parent.parent.window.blit(self.handNameText,
(self.parent.parent.width/2 - (self.handNameTextRect.width /2),
(self.parent.parent.height * 9/10) - self.handNameTextRect.height))


        fpsText =
self.font.render(str(int(self.parent.parent.clock.get_fps())), True,
self.textColour)
        if self.parent.server:
            if self.parent.server.isInternal:
                serverInfoText = self.font.render("Singleplayer", True,
self.textColour)
            else:
                serverInfoText = self.font.render("Server Ip: " +
self.parent.server.globalIp + ":" + str(self.parent.parent.client.port),
True, self.textColour)
        else:
            serverInfoText = self.font.render("Server Ip: " +
self.parent.parent.client.serverIp + ":" +
str(self.parent.parent.client.port), True, self.textColour)
        serverInfoTextRect = serverInfoText.get_rect()
        self.parent.parent.window.blit(fpsText, (0, 0))
        self.parent.parent.window.blit(serverInfoText, (0,
self.parent.parent.height - serverInfoTextRect.height))

        if self.handShown == 1:
            self.flipHand()
        keys = pygame.key.get_pressed()
        if keys[pygame.K_SPACE]: self.flipHand()
        if keys[pygame.K_ESCAPE]: self.togglePause()
        if keys[pygame.K_LALT]: self.toggleChat()
        if keys[pygame.K_p]:
self.parent.parent.client.networkQueue.put(b'7["Mr Margon"]\x1e')
        if keys[pygame.K_TAB]:
            self.pokerHelp = True
        else:
            self.pokerHelp = False
        if self.parent.parent.client.turn:
            if keys[pygame.K_UP]:
                self.parent.parent.client.increaseBet()
            elif keys[pygame.K_DOWN]:
                self.parent.parent.client.decreaseBet()
            elif keys[pygame.K_f]:
                self.parent.parent.client.betting = 0
                self.parent.parent.client.fold()

        if self.paused:
            self.pauseMenu()
        if self.chatting:
            self.chatMenu()

    def remove(self):
        for i in self.potChips:
            i.remove()
        for i in self.chips:
            i.remove()
        self.removeCommunityCards()
```

Key Algorithms and Classes – compare.py

Overview

The comparing module is a key module in my program, without this module, it would be very hard to compare other players hands and display your own. The module has two main functions, one being taking an array of cards and returning the best hand with those card as a numerical value which can be compared, and the other function converting the numerical value into a name which can be displayed on screen and read by the player so that they know what hand they have.

sortRule()

This is a very basic function used by the main 'Get Value of Hand' function used to return the value of the hand so that the cards are able to be sorted in value order by the sort() function.

Sorting Rule

```
def sortRule(x):
    return x.value
```

concatenateInts()

This again is another very basic function which is used to concatenate two number together as python has no inbuilt function for this and I need the integers to remain integers. This function is used when combing the score of the hand together. For example 03 is the code for Three of a kind and the number 09 is the code for a then therefore the code needs to be concatenated to product 0309.

Concatenation Function

```
def concatenateInts(a, b):
    return a*100 + b
```

getValueOfHand()

This is the main part of the compare module. It takes an array of card objects and figures out the best possible hand with the provided cards as a numerical value. It works by first sorting the cards in the order of lowest value to highest value using the sorting rule from earlier. Once the hand is sorted in value order, obtaining the highest value hand starts.

The value of the hand is obtained in one pass over the list therefore having the time complexity of O(n) meaning it will only take the length of time to loop over the list once.

One the first loop, the first card has no comparisons made against it and it is stored as the previous card. This is done as it is not possible to compare a single card and the code would error as the previous card value isn't set to anything.

The next cards are then immediately compared to see if the value of that card is equal to the previous card, if so then it increments a counter by one. This indicates to the algorithm that one pair has been found so far or a three or four of a kind. If concurrency counter is greater than one and the check fails, then what ever type of pair, three of a kind, or four of a kind is recorded in an array where first value indicates the number of pairs found, next is number of three of a kind found, and last is four of a kind found.

The next check is to see that if there is a straight. This is done by checking if the previous value plus one is equal to current card value, if it is so then the straight counter is incremented by one and if it is equal to four, then a straight has been found. If the straight check doesn't pass and is greater than one. Then the counter is set back to 0. Next the

simplest check is done, depending on the suit of the card, a counter is incremented to indicate the amount of same suited cards in a player's hand, if one count is equal and exceeds 5, then the player has a flush.

Next the high card is retrieved, due to the fact that the list of the player's cards are sorted in value order, the highest value card would be at the end of the list. In this case the high card is retrieved by going to the end of the list.

Lastly the algorithm figures out the best hand the player has and returns it as a numerical value. It first checks to see is there is one four of a kind, if there is, then the code for a four of a kind and the highest value card in the hand is concatenated and appended to the 'hands' list. Next the check for three of a kind is done, this is done in the same way as four of a kind, and then the check for the pair.

Next it checks for either a full house which is one pair plus a three of a kind, if found then the code for a full house plus the highest value card in the hand is appended to the hands list. Same goes for the two pair however it just checks for to see if the pair counter is greater of equal to 2.

Lastly a check is done to see if a flush is found, and or a flush high straight, then just a straight.

Finally, the algorithm returns the highest value in the hands list, being the players best hand.

Get Value of Hand Code

```
def getValueOfHand(hand):
    if len(hand) < 2:
        return 0

    # hand value = 01-10 + 01-13 e.g. 0510 = jack high straight

    hand.sort(key=sortRule)
    concurrent = 0
    straight = 0
    suitCount = [0, 0, 0, 0]
    pairCount = [0, 0, 0] # pair, three, four
    suits = ["Clubs", "Diamonds", "Hearts", "Spades"]
    prevCard = None
    hands = []
    hasStraight = False
    hasFlush = False
    highestStraightCard = None
    highestPairCard = None
    for i in range(len(hand)):
        if not i == 0:

            # check for pair, two pair, three of a kind, and four of a kind
            if prevCard.value == hand[i].getValue():
                concurrent += 1

            if concurrent >= 1 and not prevCard.getValue() ==
hand[i].getValue():
                pairCount[concurrent - 1] += 1
                concurrent = 0
                highestPairCard = prevCard
            elif not prevCard.getValue() == hand[i].getValue():
                concurrent = 0
```

```
            # check for straight - 05
            if prevCard.getValue() + 1 == hand[i].getValue():
                straight += 1

            if straight >= 4 and not prevCard.getValue() + 1 ==
hand[i].getValue():
                highestStraightCard = prevCard
                hasStraight = False
                straight = 0
            elif not prevCard.getValue() + 1 == hand[i].getValue():
                straight = 0

        prevCard = hand[i]

        #check for flush
        suitCount[suits.index(hand[i].getSuit())] += 1

    # get high card - 01
    highCard = hand[len(hand) - 1]
    value = concatenateInts(1, highCard.getValue())
    hands.append(value)
#     print(highCard.getValueName(), 'high')

    # print pairs - 02, 03, 04, 08
    for i in range(len(pairCount)):
        if i == 0:
            if pairCount[i] == 1:
                value = concatenateInts(2, highestPairCard.getValue())
                hands.append(value)
#                 print("Pair of " + highestPairCard.getValueName() + "'s")
            elif pairCount[i] == 2:
                value = concatenateInts(3, highestPairCard.getValue())
                hands.append(value)
#                 print("Two Pair " + highestPairCard.getValueName() +
"'s")
        elif i == 1:
            if pairCount[i] == 1:
                value = concatenateInts(4, highestPairCard.getValue())
                hands.append(value)
#                 print("Three of a kind " + highestPairCard.getValueName()
+ "'s")
        elif i == 2:
            if pairCount[i] == 1:
                value = concatenateInts(8, highestPairCard.getValue())
                hands.append(value)
#                 print("Four of a kind " + highestPairCard.getValueName()
+ "'s")

    # check for full house - 07
    if pairCount[0] >= 1 and pairCount[1] >= 1:
        value = concatenateInts(7, highestPairCard.getValue())
        hands.append(value)
#         print("Full house " + highestPairCard.getValueName() + " high")

    # print flush - 06
    for i in range(len(suitCount)):
        if suitCount[i] >= 5:
            for card in hand:
                if card.getSuit() == suits[i]:
                    bestCard = card
```

```
            value = concatenateInts(6, bestCard.getValue())
            hands.append(value)
            hasFlush = True
#             print(suit[i] ,'Flush')


    if hasStraight:
        if hasFlush:
            value = concatenateInts(9, highestStraightCard.getValue())
            hands.append(value)
#           print(highestStraightCard.getValueName(), 'Flush High
Straight')
        else:
            value = concatenateInts(5, highestStraightCard.getValue())
            hands.append(value)
#           print(highestStraightCard.getValueName(), 'High Straight')

    return max(hands)
```

valueToName()

This algorithm is used to obtain the hand name for a numerical value parsed into the function. The algorithm contains two lists, the hand name with question marks used to format the hand name later, and a list of all the card names and their values. If the value parsed in the algorithm is equal to 0, it will return no hand as the lowest numerical value from a hand using my system anyone can receive is 0101.

The value is then split in two, one number determining the hand name, and the other the card name. If the value of the hand is 0913, it must be the best hand in the game therefore being the royal flush, therefore the game returns 'Royal Flush'. Otherwise, the algorithm returns the hand name formatted with the highest value card.

Value to Name Code

```
def valueToName(value):
    hands = {
        1 : "? High",
        2 : "Pair of ?'s",
        3 : "Two Pair ?'s",
        4 : "Three of a Kind",
        5 : "? High Straight",
        6 : "? High Flush",
        7 : "Full House",
        8 : "Four of a Kind",
        9 : "? High Straight Flush",
    }

    cardNames = {
            1 : "2",
            2 : "3",
            3 : "4",
            4 : "5",
            5 : "6",
            6 : "7",
            7 : "8",
            8 : "9",
            9 : "10",
            10 : "Jack",
            11 : "Queen",
            12 : "King",
            13 : "Ace"
```

```
        }

    if value == 0:
        return 'No Hand'

    handNumber, cardName = int(str(value)[:1]),
cardNames[int(str(value)[1:])]

    if value != 913:
        handName = hands[handNumber].replace("?", cardName)
    else:
        handName = "Royal Flush"

    return handName
```

Key Algorithms and Classes – assetloader.py

Overview

The asset loader in my program allowed me to load and add images into the game very easily.

imageGroups

In order to specify to the program, the location and existence of the assets to the program it is essential I define it in a dictionary like I did below. This made it easy to add additional images to my program during development and reference them in game. All I had to do was to add an image was update the file path in the dictionary to the correct format as shown below.

Image Groups Dictionary

```
imgGroups = {
    "Assets/Cards/": [
            '2 of Hearts',
            '3 of Hearts',
            '4 of Hearts',
            '5 of Hearts',
            '6 of Hearts',
            '7 of Hearts',
            '8 of Hearts',
            '9 of Hearts',
            '10 of Hearts',
            'Jack of Hearts',
            'Queen of Hearts',
            'King of Hearts',
            'Ace of Hearts',
            '2 of Diamonds',
            '3 of Diamonds',
            '4 of Diamonds',
            '5 of Diamonds',
            '6 of Diamonds',
            '7 of Diamonds',
            '8 of Diamonds',
            '9 of Diamonds',
            '10 of Diamonds',
            'Jack of Diamonds',
            'Queen of Diamonds',
            'King of Diamonds',
            'Ace of Diamonds',
            '2 of Clubs',
            '3 of Clubs',
            '4 of Clubs',
```

```
            '5 of Clubs',
            '6 of Clubs',
            '7 of Clubs',
            '8 of Clubs',
            '9 of Clubs',
            '10 of Clubs',
            'Jack of Clubs',
            'Queen of Clubs',
            'King of Clubs',
            'Ace of Clubs',
            '2 of Spades',
            '3 of Spades',
            '4 of Spades',
            '5 of Spades',
            '6 of Spades',
            '7 of Spades',
            '8 of Spades',
            '9 of Spades',
            '10 of Spades',
            'Jack of Spades',
            'Queen of Spades',
            'King of Spades',
            'Ace of Spades',
            'card_back',
            'card_empty',
        ],
    "Assets/Default Pfp/": [
            "gameoil",
            "teams",
            "no image",
            "Graham",
            "Lukas",
            "Bahn",
            "Jim",
            "Jorge",
            "Kallum",
            "Sachet",
            "Red Chief",
            "John",
            "Greg",
            "Joe",
            "Margaret",
            "Barbara",
            'Agatha',
            'Helena',
            'Christine',
            'Jenny'
        ],
    "Assets/Chips/": [
            '1',
            '2',
            '4',
            '8',
            '16',
            'Big blind',
            'Little blind',
            'Dealer',
            'Poker icon',
        ],
    "Assets/Keys/": [
            'Alt',
```

```
            'Down',
            'Enter',
            'F',
            'Space',
            'Up',
            'Tab',
        ],
    "Assets/Misc/": [
            'Poker Assistance',
        ],
    "Assets/": ["Logo"]
}
```

load()

The load function goes through the imageGroups dictionary and creates a new dictionary with the index being the file name, and the value being the file path. Doing it like this made it easy to reference and update images in the game as all I had to do to load the image was to reference the dictionary with the index of the image I wanted. This also came with the benefit that all the images were loaded beforehand, not allowing the game to slow when displaying new images.

load code

```
def load():
    for (name, group) in imgGroups.items():
        for item in group:
            ext = ".png"
            splitted = item.split(".")
            if len(splitted) > 1:
                item = splitted[0]
                ext = "." + splitted[1]

            imagePaths[item] = os.path.join(name, item + ext)
```

Key Algorithms and Classes – client.py (networking)

Overview

The client networking module allows the game to communicate with servers and give the player the ability to play against other players in poker games. The client networking module communicates to the main client with information about the game so that the user is able to see the game commencing on their screen. The client networking module also allows the user to send commands to the server such as folding or betting.

packetID

This packet id list allows the client and the server to distinguish between different types of packets and the name/type of packet. This is used to figure out on the client what packet is for what purpose and allows the client to communicate accordingly.

Client Packet ID List

```
packetID = {
    '0' : 'Connect',
    '1' : 'Disconnect',
    '2' : 'Player Data',
    '3' : 'New Game',
    '4' : 'Picture',
    '5' : 'Id',
    '6' : 'Loaded',
    '7' : 'Message',
    '8' : 'Start Game',
```

```
    '9' : 'Turn',
    '10' : 'Bet',
    '11' : 'Fold',
    '1000' : 'Ping',
}
```

Client Class

The client class handles bridging the gap between the main game and the server. The client connects to the server and then constantly listens and sends information to the server. When the client receives data, the number at the start of the packet signifies the packet id and is checked to see what type of packet it is. Depending on the packet type, the client will behave differently to the packet data. For example, if the packet is a player data packet, then it will unpack the player data in the packet and prepare to organise the data so that it shows up correctly for the player. The packet also gains a \x1e character on the end of the packet so that the client knows that the entire packet has been sent instead of processing a part of sent packet and therefore erroring. The client also is able to send information to the server where it is processed and then sent to other clients connected to that server if needed.

Client Class Code

```
class Client():
    def __init__(self, username, pfp='', main=None):
        self.username = username
        self.pfp = pfp
        self.main = main

        self.client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.client.settimeout(1)
        self.connection = None

        self.turn = False
        self.betting = 0
        self.money = [0, 0, 0, 0, 0]
        self.values = [1, 2, 4, 8, 16]
        self.totalMoney = 0
        self.currentPlayer = None
        self.id = None
        self.networkQueue = queue.Queue()
        self.folded = False

        self.stopping = threading.Event()
        self.timeDebounce = 0


    def serialiseClient(self):
        data = {
            'Name' : self.username,
        }

        return json.dumps(data)

    def sendingImages(self, picture):
        try:
            with open('./User Images/' + self.pfp, 'rb') as f:
                convertedImage = base64.b64encode(f.read())
                self.client.sendall((convertedImage + b'\x1e'))
        except:
            with open(assetloader.imagePaths['no image'], 'rb') as f:
```

```
                convertedImage = base64.b64encode(f.read())
                self.client.sendall((convertedImage + b'\x1e'))

    def connect(self, serverIp, port):
        self.serverIp = serverIp
        self.port = int(port != ''  and port or 27015)
        try:
            self.client.connect((self.serverIp, self.port))
            threading.Thread(target=self.clientLoop, args=(),
daemon=True).start()
            return True
        except Exception as e:
            print("Connection error:", e)
            return False

    def send(self, data):
        try:
            self.client.sendall((data + '\x1e').encode())
        except Exception as e:
            print('Sending error:', e)

    def recieveData(self):
        data = b''
        while not self.stopping.is_set():
            try:
                recv = self.client.recv(4096)
                data += recv
                if not recv:
                    break
                elif recv[-1:] == b'\x1e':
                    data = data[:-1]
                    break
            except Exception as e:
                break
            except:
                break
        packetId = ''
        while len(data) > 0 and chr(data[0]).isnumeric():
            packetId += chr(data[0])
            data = data[1:]
        if packetId == '':
            return '1000', None
        return packetId, data

    def clientLoop(self):
        self.send(self.serialiseClient())
        self.sendingImages(self.pfp)
        while not self.stopping.is_set():
            packetId, data = self.recieveData()
            match packetID[packetId]:
                case "Connect":
                    pass
                case "Disconnect":
                    pass
                case "Id":
                    data = json.loads(data.decode())
                    self.id = data[0]
                case "Player Data":
                    print(data)
                    data = json.loads(data.decode())
                    self.main.game.players = data['Players']
```

```
                    self.turn = data['Current Player'] and (data['Current
Player']['Id'] == self.main.client.id)

                    if
len(self.main.game.mainPlayerHUD.playerDisplays.keys()) > 0:
                        for idx, val in enumerate(data['Players']):
                            if (val['Id'] != self.main.client.id):

self.main.game.mainPlayerHUD.playerDisplays[val['Id']].money = val['Money']

self.main.game.mainPlayerHUD.playerDisplays[val['Id']].action = val['Last
Action']

self.main.game.mainPlayerHUD.playerDisplays[val['Id']].username =
val['Name']

                    self.main.game.mainPlayerHUD.hand = [
                        self.main.Card(self.main, data['Hand'][0]['suit'],
data['Hand'][0]['value']),
                        self.main.Card(self.main, data['Hand'][1]['suit'],
data['Hand'][1]['value'])
                    ]
                    self.main.game.communityCards = []
                    for card in data['Community Cards']:

self.main.game.communityCards.append(self.main.Card(self.main,
card['suit'], card['value']))
                    self.main.game.mainPlayerHUD.drawHand()

                    if data['Current Player']:
                        self.currentPlayer = data['Current Player']['Name']

                    print(data)
                    self.main.client.money = data['Money']
                    self.main.game.mainPlayerHUD.role = data['Role']
                    self.main.game.pot = data['Pot']
                    self.main.game.lastbet = data['Last Bet']
                case "New Game":
                    for display in
self.main.game.mainPlayerHUD.playerDisplays.values():
                        display.remove2()
                    self.main.game.mainPlayerHUD.playerDisplays = {}
                    self.main.game.mainPlayerHUD.drawPlayers()
                    for clientId, playerInfo in
self.main.game.mainPlayerHUD.playerDisplays.items():
                        if os.path.exists('./pfp/' +  clientId + '.png'):

playerInfo.profilePicture.update(pygame.image.load('./pfp/' + clientId +
'.png'))
                    self.networkQueue.put(b'6[]\x1e')
                case "Picture":
                    data = data[1:]
                    clientId = ''
                    while len(data) > 0 and chr(data[0]).isnumeric():
                        clientId += chr(data[0])
                        data = data[1:]
                    data = data[1:]
                    picture = base64.b64decode(data + b'==')
                    with open('./pfp/' + clientId + '.png', 'wb') as f:
                        f.write(picture)
```

```
                    for cid, playerInfo in
self.main.game.mainPlayerHUD.playerDisplays.items():
                        if os.path.exists('./pfp/' +  clientId + '.png')
and cid == clientId:

playerInfo.profilePicture.update(pygame.image.load('./pfp/' + clientId +
'.png'))
                case "Message":
                    data = str(data.decode('utf-8'))
                    print(data)
                    if len(self.main.game.mainPlayerHUD.chatMessages) <= 4:

self.main.game.mainPlayerHUD.chatMessages.append(data[0])
                    else:
                        self.main.game.mainPlayerHUD.chatMessages.pop(0)

self.main.game.mainPlayerHUD.chatMessages.append(data[0])
                case "Turn":
                    self.main.client.turn = True
                    pygame.mixer.Sound("Assets/Sound/Ping.mp3").play()
                    self.totalMoney = self.calculateMoney()
                    self.betting = self.main.game.lastbet
                case "Start Game":
                    data = json.loads(data)
                    self.main.game.gameStart = data['Start Time']
                case _:
                    pass
            if not self.stopping.is_set():
                if not self.networkQueue.empty():
                    self.client.sendall(self.networkQueue.get())
                else:
                    self.client.sendall(b'1000[]\x1e')

    def minAmount(self):
        for idx, val in enumerate(self.money):
            if val >= 1:
                return self.values[idx]


    def increaseBet(self):
        minAmount = self.minAmount()
        if time.time() - self.timeDebounce >= 0.1:
            self.timeDebounce = time.time()
            if not self.betting + 1 > self.totalMoney:
                self.betting += minAmount

    def decreaseBet(self):
        minAmount = self.minAmount()
        if time.time() - self.timeDebounce >= 0.1:
            self.timeDebounce = time.time()
            if not self.betting - 1 < 0 and not self.betting - 1 <
self.main.game.getLastBet():
                self.betting -= minAmount

    def fold(self):
        self.folded = True
        self.betting = 0
        self.endGo()

    def endGo(self):
        if self.folded:
```

```
            self.networkQueue.put(b'11[]\x1e')
        else:
            self.networkQueue.put(bytes(f'10[{self.betting}]\x1e', 'utf-
8'))
        self.turn = False
        self.betting = 0

    def calculateMoney(self):
        value = 1
        totalMoney = 0
        for i in self.money:
            amount = value * i
            totalMoney += amount
            value *= 2
        self.totalMoney = totalMoney
        return self.totalMoney

    def bet(self, amount):
        starting = amount
        chipsAdded = [0, 0, 0, 0, 0]
        print(amount)
        if not amount == 0:
            for i in range(4, -1, -1):
                print(amount, self.values[i])
                if amount >= self.values[i] and self.money[i]:
                    amountOfTimes = amount // self.values[i]
                    if amountOfTimes <= self.money[i]:
                        chipsAdded[i] += amountOfTimes
                        amount -= amountOfTimes * self.values[i]
                    else:
                        chipsAdded[i] = self.money[i]
                        amount -= self.money[i] * self.values[i]
            if amount == 0:
                self.lastbet = starting
                print(chipsAdded)
                self.minusChips(chipsAdded)
                self.endGo()
        else:
            self.endGo()



    def minusChips(self, arrayChips):
        for index in range(len(self.money)):
            self.money[index] -= arrayChips[index]


    def disconnect(self):
        self.stopping.set()
        self.recieveData()
        self.send('1[Disconnect]')
        # self.client.close()
```

Key Algorithms and Classes – server.py (networking)

Overview

The server class handles connected clients to the server, and the main game logic. When a server is opened players are able to connect and play on the server. Additionally, if the client decides to play single player, the server is still created, but it is created as an internal server allowing only the person playing single player to connect and no one else.

packetID

The packet id list is the same as it is on the client, it needs to be the same and agreed upon otherwise the server and the client won't be able to process any requests from the client such as betting or folding. The server checks the packet for the packet id and responds according to the data depending on the type of packet.

Server Packet ID List

```
packetID = {
    '0' : 'Connect',
    '1' : 'Disconnect',
    '2' : 'Player Data',
    '3' : 'New Game',
    '4' : 'Picture',
    '5' : 'Id',
    '6' : 'Loaded',
    '7' : 'Message',
    '8' : 'Start Game',
    '9' : 'Turn',
    '10' : 'Bet',
    '11' : 'Fold',
    '1000' : 'Ping',
}
```

Server Class

The server class is the main part of the server, it is where the clients which are connected to the server are handled and game logic and information such as the chips in the pot and the current community cards etc is done. This is main part where the game occurs is handled.

When the server is created, certain parameters may be changed in order to change certain aspect of the server for example the port on which the server will open on and other user are able to connect through and if the server is an internal server or in other words a single player server.

If the server is internal then the server will restrict the server to work internally only by setting the allowed Ip address to 127.0.0.1 which is the internal Ip of the machine. Additionally, it only allows one connection and does not thread the user when connected to the server

Otherwise, the server will start as normal and run the server loop where the server will accept requests to connect to the server and play. If the current game has not started then it will add the player requesting to join directly to the client list ready to play, as long as the server isn't full and there is a bot to replace with a player. If not, then the client will be placed in a waiting list where they will be until the start of a new game where they can replace the position of a bot.

The server also needs to listen to every request from the client as well as send information often such as pinging the client to see if they are still connected to the server and sending them game information such as the current player, the players who are playing, their money, etc.

Not only does the server have to manage the clients, it also must be able to handle the game logic as well as checking that requests to play/make moves such as folding or betting are only processed by the server if the current player is the one who sent the command, otherwise it would be ignored.

During the game the server will loop over all the of the players and bots at least once and then progress with the game until every player has bet the same amount of money. The server then also needs to make sure that player which have folded are counted as non-participants meaning if it was meant to be their go they are skipped and when the winning player is meant to be decided they don't receive the winnings.

The server must also prevent players from attempting to bet below the current betting amount.

The game should and when all but one player has folded, making that player automatically win the pot and the server proceeding to the next game. Or when the server reaches the end of the showdown round and the players hands are compared and the winning player wins the pot, or if there are more than one player with the same winning hand then the pot is split.

After the game has ended, the server will need to restart the game and reset all of the variables such as money in the pot, the community cards, and most importantly redistribute the cards to the players.

Server Class Code

```
class Server():
    def __init__(self, game=None, port=27015, maxClients=5,
isInternal=False):
        self.hostname = socket.gethostname()
        self.globalIp = socket.gethostbyname(self.hostname)
        if isInternal:
            self.acceptedIp = "127.0.0.1"
        else:
            self.acceptedIp = "0.0.0.0"
        self.port = port
        self.maxClients = maxClients
        self.clients = []
        self.waitingRoom = []
        self.currentPlayer = None
        self.startDelay = 10
        self.gameStarted = False
        self.isInternal = isInternal
        self.game = game

        self.gameNumber = 0
        self.playerNumber = 0
        self.communityCards = []
        self.pot = [0, 0, 0, 0, 0]
        self.lastbet = 0
        self.currentPlayer = None
        self.currentDealer = 0
        self.turnWait = 20.0

        self.stopping = threading.Event()


        self.deck = Deck()
        self.timer = None

        self.server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        try:
            self.server.settimeout(1)
```

```
            self.server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR,
1)
            self.server.bind((self.acceptedIp, int(self.port)))
        except socket.error as e:
            print('Error at server binding:', e)

        self.server.listen(self.maxClients + (self.maxClients - 1))
        print('The bluetooth device is-uh ready to pair.')

        threading.Thread(target=self.serverLoop, args=(),
daemon=True).start()

    def recievingImages(self, client):
        try:
            data = b''
            while not self.stopping.is_set():
                try:
                    recv = client.recv(4096)
                    data += recv
                    if not recv:
                        break
                    elif recv[-1:] == b'\x1e':
                        data = data[:-1]
                        break
                except Exception as e:
                    print('Error when recieving an image:', e)
            return data
        except:
            print('hell')

    def sendingImages(self, player):
        try:
            allClients = self.clients + self.waitingRoom
            for client in allClients:
                if (not client.isBot()) and client != player:
                    client.addToQueue((b'4|'+ str(player.id).encode() +
b'|' + player.pfp + b'\x1e'))
        except:
            print('hell')

    def sendAllImages(self, player):
        try:
            allClients = self.clients + self.waitingRoom
            for client in allClients:
                if client != player:
                    player.addToQueue((b'4|'+ str(client.id).encode() +
b'|' + client.pfp + b'\x1e'))
        except:
            print('hell')

    def startGame(self):
        self.serverStartTime = time.time()

        time.sleep(self.startDelay)
        self.gameStarted = True

        self.startOfNewRound()

    def networkLoop(self):
        while not self.stopping.is_set():
            for client in self.clients + self.waitingRoom:
```

```
                if not client.isBot() and client.connected:
                    try:
                        if client.queue.qsize() > 0:
                            jargon = client.queue.get()
                            client.connection.sendall(jargon)
                    except:
                        pass
                    data = self.recieveData(client.connection, load=False)
                    packetId = ''
                    while len(data) > 0 and data[0].isnumeric():
                        packetId += data[0]
                        data = data[1:]
                    if packetId == '':
                        packetId = '1000'
                    match packetID[packetId]:
                        case "Disconnect":
                            client.connected = False
                            self.players -= 1
                            if client in self.waitingRoom:
                                self.waitingRoom.remove(client)
                            else:
                                botName = getBotName()
                                bot = ServerBot(username='Bot ' + botName,
pfp=botImages[botName], server=self)
                                if self.gameStarted:
                                    client.folded = True
                                    client.lastAction = 'Folded'
                                    bot.fold()
                                    bot.id = client.id
                                    bot.hand = [Card(suit='Spades',
value=1)] * 2
                                    # bot.money = [10, 5, 3, 2, 1]
                                    bot.lastAction = 'Sat Out'
                                self.clients[self.clients.index(client)] =
bot
                                if self.gameStarted and self.currentPlayer
and self.currentPlayer.id == client.id:
                                    self.endGo()
                            client.connection.close()
                            if self.players <= 0:
                                self.close()
                        case "Loaded":
                            if self.gameStarted:
                                client.isLoaded = True
                        case "Fold":
                            if self.gameStarted:
                                if self.currentPlayer.id == client.id:
                                    client.fold()
                                    client.lastAction = 'Folded'
                                    self.endGo()
                        case "Bet":
                            money = json.loads(data)[0]
                            if self.gameStarted:
                                if self.currentPlayer.id == client.id:
                                    if self.lastbet < money:
                                        client.betting = money
                                        client.lastAction = 'Raise: ' +
str(money)
                                        self.endGo()
                                    elif self.lastbet == money:
```

```
                                        client.lastAction = 'Call: ' +
str(money)
                                        client.betting = money
                                        self.endGo()
                                    elif client.money == self.lastbet and
self.lastbet == money:
                                        client.lastAction = 'All in'
                                        client.betting = client.money
                                        self.endGo()
                            case "Message":
                                for c in self.clients:
                                    if not c.isBot(): # client.id != c.id and
                                        c.addToQueue(b'7' + bytes(data, 'utf-
8') + b'\x1e')
                            case _:
                                pass

    def serverLoop(self):
        self.bots = 0
        self.players = 0

        for i in range(self.maxClients):
            botName = getBotName()
            self.clients.append(ServerBot(username='Bot ' + botName,
pfp=botImages[botName], server=self))
            self.bots += 1
        # try:
        threading.Thread(target=self.networkLoop, args=(),
daemon=True).start()
        if not self.isInternal:

            while not self.stopping.is_set():
                try:
                    connection, address = self.server.accept()
                    connection.settimeout(1)
                    playerInfo = self.recieveData(connection)
                    playerImage = self.recievingImages(connection)
                    print('The bluetooth device is a connected-uh
successfulay to:', address)
                    if not self.gameStarted:
                        client = ServerClient(connection, address[0],
username=playerInfo['Name'], pfp=playerImage)
                        self.clients[self.players] = client
                        self.players += 1
                        self.bots -= 1
                        self.send(client, '5' + json.dumps([client.id]))
                        if client.isHost():
                            threading.Thread(target=self.startGame,
args=(), daemon=True).start()

                        self.sendingImages(client)
                        self.sendAllImages(client)
                    elif len(self.waitingRoom) <= self.bots:
                        self.waitingRoom.append(ServerClient(connection,
address[0], username=playerInfo['Name'], pfp=playerImage))
                    else:
                        connection.close()
                except Exception as e:
                    print(e)
        else:
            connection, address = self.server.accept()
```

```
            connection.settimeout(1)
            playerInfo = self.recieveData(connection)
            playerImage = self.recievingImages(connection)
            print('The bluetooth device is a connected-uh successfulay
to:', address)
            client = ServerClient(connection, address[0],
username=playerInfo['Name'], pfp=playerImage)
            self.clients[self.players] = client
            self.players += 1
            self.bots -= 1
            self.send(client, '5' + json.dumps([client.id]))
            threading.Thread(target=self.startGame, args=(),
daemon=True).start()
            self.sendingImages(client)
            self.sendAllImages(client)
        # except KeyboardInterrupt:
            #self.close()


    def serialiseClients(self):
        data = []

        for client in self.clients:
            data.append({
                'Role' : client.role,
                'Host' : client.host,
                'Bot' : client.bot,
                'Turn' : client.turn,
                'Name' : client.username,
                'Money' : client.money,
                'Id' : client.id,
                'Folded' : client.folded,
                'Last Action' : client.lastAction,
            })

        return data

    def sendAllClients(self, data):
        try:
            allClients = self.clients + self.waitingRoom
            for client in allClients:
                if not client.isBot():
                    client.addToQueue((data + '\x1e').encode())
        except:
            print('hell')

    def recieveData(self, connection, load=True):
        data = ''
        while not self.stopping.is_set():
            try:
                recv = connection.recv(4096)
                data += recv.decode()
                if not recv:
                    break
                elif recv[-1:] == b'\x1e':
                    data = data[:-1]
                    break
            except Exception as e:
                print('General recieving error:', e)
                break
        if load:
```

```
            print(data)
            try:
                return json.loads(data)
            except:
                return ''
        else:
            return data

# game logic/functionality

    def startOfNewRound(self):

        self.gameNumber += 1
        self.playerNumber = self.currentDealer
        self.lastbet = 0
        self.communityCards = []

        for idx, client in enumerate(self.clients):
            if client.isBot() and len(self.waitingRoom) > 0:
                self.clients[idx] = self.waitingRoom.pop(0)

        for client in self.clients:
            client.setPlayingStatus(True)
            client.setRole('Player')
            client.lastbet = 0
            client.handValue = 0
            client.folded = False
            client.hand = []
            client.lastAction = ''
            client.betting = 0
            client.turn = False
            if not client.isBot():
                client.isLoaded = False
        self.clients[(self.playerNumber) %
len(self.clients)].setRole('Dealer')
        self.clients[(self.playerNumber + 1) %
len(self.clients)].setRole('Little Blind')
        self.clients[(self.playerNumber + 1) % len(self.clients)].betting =
1
        self.clients[(self.playerNumber + 1) % len(self.clients)].lastbet =
1
        self.lastbet = 1
        betAmount = self.clients[(self.playerNumber + 1) %
len(self.clients)].bet(self.lastbet)
        self.addChipsToPot(betAmount)
        self.clients[(self.playerNumber + 2) %
len(self.clients)].setRole('Big Blind')
        self.clients[(self.playerNumber + 2) % len(self.clients)].betting =
2
        betAmount = self.clients[(self.playerNumber + 2) %
len(self.clients)].bet(self.lastbet)
        self.clients[(self.playerNumber + 2) % len(self.clients)].lastbet =
2
        self.lastbet = 2
        self.addChipsToPot(betAmount)
        whoBet = (self.playerNumber + 2) % len(self.clients)#
        basebetNumber = self.lastbet

        self.playerNumber = (self.playerNumber + 3) % len(self.clients)

        self.deck.regenDeck()
```

```
        for client in self.clients:
            client.hand = [self.deck.getCard(), self.deck.getCard()]

        self.updatePlayers()

        self.sendAllClients("3[]")

        self.waitForPlayers()
        self.sendAllClients('8' + json.dumps({
            'Start Time' : time.time()
        }))
        # pre flop, flop, the turn, the river, the showdown
        endGame = False
        for i in range(5):
            if endGame:
                break

            everyoneHasNotBet = True
            while everyoneHasNotBet:
                everyoneHasNotBet = False
                for i in range(len(self.clients)):
                    if basebetNumber != 0 and whoBet and self.playerNumber
== whoBet:
                        everyoneHasNotBet = False
                        self.lastbet = 0
                        basebetNumber = 0
                        whoBet = None
                        break
                    print('Next Go')
                    self.currentPlayer = self.clients[self.playerNumber]
                    self.updatePlayers()
                    if not self.currentPlayer.folded:
                        self.timer = threading.Timer(self.turnWait,
self.endGo, args=(True,))
                        self.timer.start()
                        self.currentPlayer.promptAction()
                        self.timer.join()

                    if not self.currentPlayer.lastbet == basebetNumber and
not self.currentPlayer.folded:
                        everyoneHasNotBet = True
                        basebetNumber = self.currentPlayer.lastbet
                        whoBet = self.playerNumber
                        print('WHO BET:', whoBet, basebetNumber)
                    self.playerNumber = (self.playerNumber + 1) %
len(self.clients)
                    foldCount = 0
                    for player in self.clients:
                        if player.folded:
                            foldCount += 1
                    if foldCount + 1 >= len(self.clients):
                        endGame = True
                        everyoneHasNotBet = False
                        break
            whoBet = None
            basebetNumber = 0
            for player in self.clients:
                player.lastbet = 0
            self.lastbet = 0
```

```
            self.showCommunityCards()
        winners = self.returnWinners()
        self.rewardPlayers(winners)
        self.pot = [0, 0, 0, 0, 0]
        time.sleep(5)
        self.startOfNewRound()

    def rewardPlayers(self, winners):
        potMoney = self.calculatePot()
        reward = potMoney // len(winners)
        for winner in winners:
            winner.addChips(reward)
            winner.lastAction = 'Won: ' + str(reward) + ' with ' +
compare.valueToName(winner.handValue)

    def calculatePot(self):
        value = 1
        totalMoney = 0
        for i in self.pot:
            amount = value * i
            totalMoney += amount
            value *= 2
        return totalMoney


    def returnWinners(self):
        for player in self.clients:
            if not player.folded:
                player.handValue = compare.getValueOfHand(player.hand +
self.communityCards)
            else:
                player.handValue = 0
        maxHandValue = max(self.clients, key=lambda c:
c.handValue).handValue
        print('best hand ever', maxHandValue)
        winners = list(filter(lambda c: c.handValue >= maxHandValue and not
c.folded, self.clients))
        print('these people have won', winners)
        return winners


    def updatePlayers(self):
        clientData = self.serialiseClients()
        for client in self.clients:
            self.send(client, '2' + json.dumps({
                'Hand' : [{
                    'suit' : client.hand[0].suit,
                    'value' : client.hand[0].value,
                },
                {
                    'suit' : client.hand[1].suit,
                    'value' : client.hand[1].value,
                }],
                'Money' : client.money,
                'Role' : client.role,
                'Pot' : self.pot,
                'Players' : clientData,
                'Community Cards' : self.serialiseCommunityCards(),
                'Last Bet' : self.lastbet,
                'Current Player' : (self.currentPlayer and {
                        'Name' : self.currentPlayer.getUsername(),
```

```
                    'Id' : self.currentPlayer.id,
                }) or False,
        }))

    def serialiseCommunityCards(self):
        serialisedCards = []
        for card in self.communityCards:
            cardData = {
                'suit' : card.suit,
                'value' : card.value,
            }
            serialisedCards.append(cardData)
        return serialisedCards


    def endGo(self, outOfTime=False):
        if self.timer:
            self.timer.cancel()
        if outOfTime:
            self.currentPlayer.fold()
        elif not self.currentPlayer.folded:
            lastbet = self.currentPlayer.betting
            chipsToPot = self.currentPlayer.bet(self.lastbet)
            self.currentPlayer.lastbet = lastbet
            self.lastbet = lastbet
            print(chipsToPot)
            self.addChipsToPot(chipsToPot)

        self.updatePlayers()

    def addChipsToPot(self, arrChips):
        for idx, val in enumerate(arrChips):
            print(val)
            self.pot[idx] += val

    def waitForPlayers(self):
        for client in self.clients:
            if not client.isBot():
                while not client.isLoaded:
                    pass

    def send(self, client, data):
        try:
            if not client.isBot():
                client.addToQueue((data + '\x1e').encode())
        except Exception as e:
            print(e)

    def finishGo(self):
        self.playerNumber = (self.playerNumber + 1) % len(self.clients)
        self.currentPlayer = self.clients[self.playerNumber]
        if not self.currentPlayer.folded:
            self.currentPlayer.promptAction()
        else:
            self.finishGo(self)

    def showCommunityCards(self):
        if len(self.communityCards) == 0:
            for i in range(3):
                self.communityCards.append(self.deck.getCard())
        elif len(self.communityCards) == 3:
```

```
            self.communityCards.append(self.deck.getCard())
        elif len(self.communityCards) == 4:
            self.communityCards.append(self.deck.getCard())
        else:
            print('Unable to show more community cards')
        print(self.communityCards)

    def close(self):
        for client in self.clients + self.waitingRoom:
            if not client.isBot():
                client.connection.close()
        self.stopping.set()
```

Server Bot Class

The server bot is a basic player class with all the variable and functions needed for a player on the server to operate. When a server is created, five bots are automatically created and are ready to play, however if a player joins, then one bot will be removed from the player list and be replaced with the new connected client. The bots upon creation will have a random name and for that name and associated profile picture so that the bots feel more unique in the game. The server bot class has a few functions which are key for the game to work and operate.

For example the addChips() and minusChips() functions are used by the server to add and remove chips. This would be done when the client is betting and the server needs to update the client with the new number of chips now that the bot has bet, this is done by first referencing the bot and then calling the minusChips function and parsing in an array of chips to be removed for example [1, 0, 1, 0, 0] which equals the value 5.

The addChips function is the same but instead of removing chips from the bot, it adds them. This is used when the bot has won the game and needs to be credited the chips in the pot.

The bet function on the bot figures out if the amount of money that the bot would like to bet is possible and if so set the betting amount to the value and end the current go which in turn put the money in the pot and steps the turn forward.

The promptAction() function is used by the server to prompt the bot to make their turn, where they will wait a random amount of time to make the game more fluid rather than the bot making its decision instantly and confusing the player. The prompt action function check whether the community card have been revealed, and if not then it will user the winning percentages table to figure out the strength of their hand. Additionally, the bot will take into consideration its confidence level which is random per bot and is treated as a number between 0 and 1. Depending on these factors and money the bot may decide to call, fold, or raise a variable amount depending on the strength of their hand and money.

If there are community cards revealed, then the bot will take into consideration its overall hand plus other factors from before to make decisions on whether to fold or call etc.

The table for winning percentages used by the bot is below. It is a dictionary of dictionaries. Making it like this made it easy to get the percentage from the table. For example, in order to get the starting hand percentage all I had to do was reference the table and index it with the value of the two cards for example winningPercentages[7][12] would return 0.58.

Server Bot Starting Hand Probabilities Table

```
winningPercentages = {
    13 : {
```

```
        13 : 0.85,
        12 : 0.68,
        11 : 0.67,
        10 : 0.66,
        9 : 0.66,
        8 : 0.64,
        7 : 0.63,
        6 : 0.63,
        5 : 0.62,
        4 : 0.62,
        3 : 0.61,
        2 : 0.60,
        1 : 0.59,
    },
    12 : {
        13 : 0.66,
        12 : 0.83,
        11 : 0.64,
        10 : 0.64,
        9 : 0.63,
        8 : 0.61,
        7 : 0.60,
        6 : 0.59,
        5 : 0.58,
        4 : 0.58,
        3 : 0.57,
        2 : 0.56,
        1 : 0.55,
    },
    11 : {
        13 : 0.65,
        12 : 0.62,
        11 : 0.80,
        10 : 0.61,
        9 : 0.61,
        8 : 0.59,
        7 : 0.58,
        6 : 0.56,
        5 : 0.55,
        4 : 0.55,
        3 : 0.54,
        2 : 0.53,
        1 : 0.52,
    },
    10 : {
        13 : 0.65,
        12 : 0.62,
        11 : 0.59,
        10 : 0.78,
        9 : 0.59,
        8 : 0.57,
        7 : 0.56,
        6 : 0.54,
        5 : 0.53,
        4 : 0.52,
        3 : 0.51,
        2 : 0.50,
        1 : 0.50,
    },
    9 : {
        13 : 0.64,
```

```
        12 : 0.61,
        11 : 0.59,
        10 : 0.57,
        9 : 0.75,
        8 : 0.56,
        7 : 0.54,
        6 : 0.53,
        5 : 0.51,
        4 : 0.49,
        3 : 0.49,
        2 : 0.48,
        1 : 0.47,
    },
    8 : {
        13 : 0.62,
        12 : 0.59,
        11 : 0.57,
        10 : 0.55,
        9 : 0.53,
        8 : 0.72,
        7 : 0.53,
        6 : 0.51,
        5 : 0.50,
        4 : 0.48,
        3 : 0.46,
        2 : 0.46,
        1 : 0.45,
    },
    7 : {
        13 : 0.61,
        12 : 0.58,
        11 : 0.55,
        10 : 0.53,
        9 : 0.52,
        8 : 0.50,
        7 : 0.69,
        6 : 0.50,
        5 : 0.49,
        4 : 0.47,
        3 : 0.45,
        2 : 0.43,
        1 : 0.43,
    },
    6 : {
        13 : 0.60,
        12 : 0.57,
        11 : 0.54,
        10 : 0.52,
        9 : 0.50,
        8 : 0.48,
        7 : 0.47,
        6 : 0.67,
        5 : 0.48,
        4 : 0.46,
        3 : 0.45,
        2 : 0.43,
        1 : 0.41,
    },
    5 : {
        13 : 0.59,
        12 : 0.56,
```

```
        11 : 0.53,
        10 : 0.50,
        9 : 0.48,
        8 : 0.47,
        7 : 0.46,
        6 : 0.45,
        5 : 0.64,
        4 : 0.46,
        3 : 0.44,
        2 : 0.42,
        1 : 0.40,
    },
    4 : {
        13 : 0.60,
        12 : 0.55,
        11 : 0.52,
        10 : 0.49,
        9 : 0.47,
        8 : 0.45,
        7 : 0.44,
        6 : 0.43,
        5 : 0.43,
        4 : 0.61,
        3 : 0.44,
        2 : 0.43,
        1 : 0.41,
    },
    3 : {
        13 : 0.59,
        12 : 0.54,
        11 : 0.51,
        10 : 0.48,
        9 : 0.46,
        8 : 0.43,
        7 : 0.42,
        6 : 0.41,
        5 : 0.41,
        4 : 0.41,
        3 : 0.58,
        2 : 0.43,
        1 : 0.40,
    },
    2 : {
        13 : 0.58,
        12 : 0.54,
        11 : 0.50,
        10 : 0.48,
        9 : 0.45,
        8 : 0.43,
        7 : 0.40,
        6 : 0.39,
        5 : 0.39,
        4 : 0.39,
        3 : 0.38,
        2 : 0.55,
        1 : 0.39,
    },
    1 : {
        13 : 0.57,
        12 : 0.53,
        11 : 0.49,
```

```
        10 : 0.47,
        9 : 0.44,
        8 : 0.42,
        7 : 0.40,
        6 : 0.37,
        5 : 0.37,
        4 : 0.37,
        3 : 0.36,
        2 : 0.35,
        1 : 0.51,
    },
}
```

Server Bot Class Code

```
class ServerBot():
    def __init__(self, username='Bot', pfp=None, server=None):
        global Id
        self.isPlaying = False
        self.role = 'Player'
        self.host = False
        self.bot = True
        self.turn = False
        self.pfp = pfp
        self.username = username
        self.money = [10, 5, 3, 2, 1]
        self.hand = None
        self.picture = None
        self.betting = 0
        self.overallMoney = self.calculateMoney()
        self.folded = False
        self.id = str(Id)
        self.lastbet = 0
        self.server = server
        self.handValue = 0
        self.lastAction = ''
        Id += 1

        self.botConfidence = random.randint(20, 90)/100


    def isHost(self):
        return self.host

    def setPlayingStatus(self, status):
        self.isPlaying = status

    def setRole(self, role):
        self.role = role

    def isBot(self):
        return self.bot

    def setTurn(self, value):
        self.turn = value

    def getTurn(self):
        return self.turn

    def promptAction(self):
        # time.sleep(random.randint(2, 10))
```

```
        self.folded = False
        moneyRatio = 1.5 * self.calculateMoney()/64
        bettingWinChance =
winningPercentages[self.hand[0].value][self.hand[1].value] /
self.botConfidence * moneyRatio
        print(bettingWinChance)

        if len(self.server.communityCards) < 3:
            if random.random() < bettingWinChance:
                if bettingWinChance > random.randint(80 - round(5 *
self.botConfidence), 80 + round(5 * self.botConfidence))/100:
                    self.betting = self.server.lastbet + random.randint(1,
round((5 * moneyRatio * self.botConfidence/100)))
                    print(self.username, 'i am betting this')
                else:
                    self.betting = self.server.lastbet
                    print(self.username, 'i am betting this')
            else:
                self.fold()
                print(self.username, 'i fold')
        else:
            handValueWinning = compare.getValueOfHand(self.hand +
self.server.communityCards)/100 / self.botConfidence + bettingWinChance
            if random.random() < handValueWinning:
                if handValueWinning > random.randint(80 - round(5 *
self.botConfidence), 80 + round(5 * self.botConfidence))/100:
                    self.betting = self.server.lastbet + random.randint(1,
round(15 * moneyRatio * self.botConfidence/100))
                    print(self.username, 'i am betting this')
                else:
                    self.betting = self.server.lastbet
                    print(self.username, 'i am betting this')
            else:
                self.fold()
                print(self.username, 'i fold')
        self.server.endGo()

    def fold(self):
        self.lastAction = 'Folded'
        self.folded = True


    def setGame(self, game):
        self.currentGame = game

    def getUsername(self):
        return self.username

    def getHand(self):
        return self.hand

    def getRole(self):
        return self.role

    def getMoney(self):
        return self.money

    def bet(self, lastbet):
        chipsAdded = [0, 0, 0, 0, 0]
        values = [1, 2, 4, 8, 16]
        if lastbet < self.betting:
```

```
            self.lastAction = 'Raise: ' + str(self.betting)
        elif lastbet == self.betting:
            self.lastAction = 'Call: ' + str(self.betting)
        elif self.betting == self.overallMoney:
            self.lastAction = 'All in'
        elif self.betting == 0:
            self.lastAction = 'Check'
        if not self.betting == 0:
            for i in range(4, -1, -1):
                print(self.betting, values[i])
                if self.betting >= values[i] and self.money[i]:
                    amountOfTimes = self.betting // values[i]
                    if amountOfTimes <= self.money[i]:
                        chipsAdded[i] += amountOfTimes
                        self.betting -= amountOfTimes * values[i]
                    else:
                        chipsAdded[i] = self.money[i]
                        self.betting -= self.money[i] * values[i]
        elif self.betting == 0:
            self.betting = 0
            print(chipsAdded)
        self.minusChips(chipsAdded)
        return chipsAdded

    def minusChips(self, arrayChips):
        for index in range(len(self.money)):
            self.money[index] -= arrayChips[index]

    def addChips(self, money):
        values = [1, 2, 4, 8, 16]
        if not money == 0:
            for i in range(4, -1, -1):
                if money >= values[i]:
                    amountOfTimes = money // values[i]
                    if amountOfTimes > 5 and values[i] > 4:
                        amountOfTimes = min([amountOfTimes, 5])
                    self.money[i] += amountOfTimes
                    money -= amountOfTimes * values[i]


    def calculateMoney(self):
        value = 1
        money = 0
        for i in self.money:
            amount = value * i
            money += amount
            value *= 2
        self.overallMoney = money
        return money
```

Server Client Class

The server client class inherits from the server bot class making them almost identical as both need the same functions and variables. Making the class inherit from the server bot class was beneficial, as it means any information I added to the bot also made it to the client. This class is needed to keep track of player on the server including their money and hand etc

However, I did need to change a few variables and functions in my class in order for it to work.

The first thing that needed adding and probably the most important was adding under the client class was the client's connection and address. If this wasn't added then it would have been impossible to contact the player associated with the client and send them information such as the game information.

Also a few functions needed to be added or overwritten, the most notable of which being the prompt action, as we needed the client to make their action and not a computer. Overwriting this function with a function which pings the player with a packet instructing the player that it is their turn is way to make it so that the promptAction() function allows the client able to play.

Additionally, a network function was added to the server. This is to add the packets which will be sent to the client on a network queue where the server can then send the packet which is at the start of the queue to the player.

Server Client Class Code

```
class ServerClient(ServerBot):
    def __init__(self, connection, address, username='Mohaves', pfp=None):
        super().__init__(username, pfp)
        self.connection = connection
        self.address = address
        self.bot = False
        self.queue = queue.Queue()
        self.isLoaded = False
        self.betting = 0
        self.connected = True

        if self.address == "127.0.0.1":
            self.host = True
        else:
            self.host = False

    def addToQueue(self, data):
        self.queue.put(data)

    def promptAction(self):
        self.addToQueue(b'9[]\x1e')
```

Deck Class

The deck class is a crucial part of the game logic which is needed for the server to hand out cards as well as obtain the community cards for the game. The deck contains an array of card objects on the server which can be obtained with the getCard() function. This function picks a random card in the deck and then pops that card of the queue so that it cant be selected again, allowing for duplicate cards. The regen deck function just regenerates the entire deck so that there are no missing cards and they are able to be selected again.

Deck Class Code

```
class Deck():
    def __init__(self):
        self.suits = ["Clubs", "Diamonds", "Hearts", "Spades"]
        self.regenDeck()

    def getCard(self):
        self.choice = random.choice(self.deck)
        self.deck.pop(self.deck.index(self.choice))
        return self.choice
```

```
    def regenDeck(self):
        self.deck = []
        for suit in self.suits:
            for value in range(1, 14):
                self.deck.append(Card(suit, value))
```

Card Class

The card class here is almost identical to the one on the client. The only differences being no rendering functions this is because the cards are on the server. The cards on the server need to almost identical to the cards on the client so that data can be transferred correctly.

Card Class Code

```
class Card():
    def __init__(self, suit, value):
        self.suit = suit
        self.value = value
        self.values = {
            1 : "2",
            2 : "3",
            3 : "4",
            4 : "5",
            5 : "6",
            6 : "7",
            7 : "8",
            8 : "9",
            9 : "10",
            10 : "Jack",
            11 : "Queen",
            12 : "King",
            13 : "Ace"
        }
        if self.value:
            self.name = self.values[self.value] + " of " + self.suit
        else:
            self.name = None

        self.cardShown = False

    def getValue(self):
        return self.value

    def setValue(self, nValue):
        self.value = nValue

    def getName(self):
        return self.name

    def getValueName(self):
        return self.values[self.value]

    def setName(self):
        self.name = self.values[self.value] + " of " + self.suit

    def getSuit(self):
        return self.suit

    def setSuit(self, nSuit):
        self.suit = nSuit
```

Objectives

13. Main Menu

13.1. The user of the program upon opening the game, should be greeted with the main options of the game which include a play button, a settings button, and a quit button.

```
def mainMenu(self):
        self.removeAllSprites()
        self.background = self.grey
        self.logo = Image(self, xy=[self.width/2 - 84, 30], dim=[169,95],
group=self.sprites, image=self.images['Logo'])
        self.b1 = Button(self, xy=[self.width/2, 150],
function=self.modeSelect, text='Play', dim=[200, 50], group=self.objects)
        self.b2 = Button(self, xy=[self.width/2, 200],
function=self.settingsMenu, text='Settings', dim=[200, 50],
group=self.objects)
        self.b3 = Button(self, xy=[self.width/2, 250], function=self.quit,
text='Quit', dim=[200, 50], group=self.objects)
        self.qr = Image(self, xy=[self.width/2, 350], dim=[58,58],
group=self.sprites, image=self.images['QR'])
```

13.2. User of the program should be able to select the play button.

13.2.1. Upon clicking this button, the menu will be updated to show new buttons relating to the button the user just pressed, this being the play button. Users will be asked using buttons whether they would like to play single player or multiplayer or return to the main menu.

```
def modeSelect(self):
        self.removeAllSprites()
        self.background = self.grey
        self.logo = Image(self, xy=[self.width/2 - 84, 30], dim=[169,95],
group=self.sprites, image=self.images['Logo'])
        self.b1 = Button(self, xy=[self.width/2, 150],
function=self.singleplayer, text='Singleplayer', dim=[200, 50],
group=self.objects)
        self.b2 = Button(self, xy=[self.width/2, 200],
function=self.joinOrCreate, text='Multiplayer', dim=[200, 50],
group=self.objects)
        self.b3 = Button(self, xy=[self.width/2, 250],
function=self.mainMenu, text='Back', dim=[200, 50], group=self.objects)
        self.qr = Image(self, xy=[self.width/2, 350], dim=[58,58],
group=self.sprites, image=self.images['QR'])
```

13.2.1.1. Clicking the single player button should start a single player session with 4 other bots who will play along with the player until the user decides to quit.

```
    def singleplayer(self):
        self.removeAllSprites()
        self.background = self.green
        self.game = Game(self, gametype="Singleplayer")
```

13.2.1.2. Clicking the multiplayer button will send the user to a new menu with buttons asking if the user would like to join a server or create one as well as return to the previous menu.

```
def joinOrCreate(self):
        self.removeAllSprites()
        self.background = self.grey
        self.logo = Image(self, xy=[self.width/2 - 84, 30], dim=[169,95],
group=self.sprites, image=self.images['Logo'])
        self.b1 = Button(self, xy=[self.width/2, 150],
function=self.joinServerMenu, text='Join a lobby', dim=[200, 50],
group=self.objects)
        self.b2 = Button(self, xy=[self.width/2, 200],
function=self.createMultiplayerServer, text='Create a lobby', dim=[200,
50], group=self.objects)
        self.b3 = Button(self, xy=[self.width/2, 250],
function=self.modeSelect, text='Back', dim=[200, 50], group=self.objects)
        self.qr = Image(self, xy=[self.width/2, 350], dim=[58,58],
group=self.sprites, image=self.images['QR'])
```

13.2.1.2.1. Clicking the join lobby button will send the user to another menu where they will be prompted to enter the server ip as well as the server port and then be able to click 'Join' to join that server.

```
def joinServerMenu(self):
        self.removeAllSprites()
        self.background = self.grey
        self.userText = TextLabel(self, text='Server Ip:', xy=[self.width/2
- 185, 150], group=self.objects)
        self.portText = TextLabel(self, text='Port:', xy=[self.width/2 -
150, 186], group=self.objects)
        self.t1 = TextInput(self, default='', xy=[self.width/2 - 100, 150],
group=self.objects)
        self.t2 = TextInput(self, default='', xy=[self.width/2 - 100, 186],
group=self.objects)
        self.b1 = Button(self, xy=[self.width/2, 250],
function=self.connectToServer, text='Join', dim=[200, 50],
group=self.objects)
        self.b2 = Button(self, xy=[self.width/2, 300],
function=self.joinOrCreate, text='Back', dim=[200, 50], group=self.objects)
```

13.2.1.2.2. Clicking create server will create a server hosting a game for the user and connect that user, then allowing other users to enter the game.

```
    def createMultiplayerServer(self):
        self.removeAllSprites()
        self.background = self.green
        self.game = Game(self, gametype="Multiplayer - C")
```

13.2.1.2.3. Previous button should return the user back to the previous menu.

13.3. User of the program should be able to select the settings button.

13.3.1. User will be shown settings that they can change such as username, default port on which servers will be made, full screen, and screen size.

13.3.1.1. Toggleable settings are a button.

13.3.1.2. Text boxes, where written inputs are entered, pressing enter will save the information.

```
def settingsMenu(self):
        self.removeAllSprites()
        self.background = self.grey
        self.userText = TextLabel(self, text='Username:', xy=[self.width/2
- 180, 150], group=self.objects)
        self.pictureText = TextLabel(self, text='Picture:',
xy=[self.width/2 - 160, 186], group=self.objects)
        self.portText = TextLabel(self, text='Port:', xy=[self.width/2 -
140, 222], group=self.objects)
        self.widthText = TextLabel(self, text='Width:', xy=[self.width/2 -
150, 318], group=self.objects)
        self.heightText = TextLabel(self, text='Height:', xy=[self.width/2
- 150, 354], group=self.objects)
        self.t1 = TextInput(self, default=settings['Username'],
xy=[self.width/2 - 100, 150], function=self.writeSettings,
param='Username', group=self.objects)
        self.t2 = TextInput(self, default=settings['Picture'],
xy=[self.width/2 - 100, 186], function=self.writeSettings, param='Picture',
group=self.objects)
        self.t3 = TextInput(self, default=settings['Server Port'],
xy=[self.width/2 - 100, 222], function=self.writeSettings, param='Server
Port', group=self.objects)
        self.b1 = Button(self, xy=[self.width/2, 287],
function=self.updateFullscreen, text='Fullscreen Toggle' , dim=[250, 50],
group=self.objects)
        self.fScreentxt = TextLabel(self, text='(restart to take effect)',
xy=[self.width/2 + 290, 272], group=self.objects)
        self.t4 = TextInput(self, default=settings['Width'],
xy=[self.width/2 - 100, 318], function=self.writeSettings, param='Width',
group=self.objects)
        self.t5 = TextInput(self, default=settings['Height'],
xy=[self.width/2 - 100, 354], function=self.writeSettings, param='Height',
group=self.objects)
        self.b2 = Button(self, xy=[self.width/2, 419],
function=self.mainMenu, text='Back', dim=[200, 50], group=self.objects)
```

13.4. User of the program should be able to select the quit button.

13.4.1. Upon clicking select the game should close.

```
    def quit(self):
        self.running = False
        self.client.disconnect()
```

14. Single player

14.1. If a single player game is made, then the game will fill the other four player slots with bots so that the player is able to play against opponents.

```
connection, address = self.server.accept()
connection.settimeout(1)
playerInfo = self.recieveData(connection)
playerImage = self.recievingImages(connection)
```

```
print('The bluetooth device is a connected-uh successfulay to:', address)
client = ServerClient(connection, address[0], username=playerInfo['Name'],
pfp=playerImage)
self.clients[self.players] = client
self.players += 1
self.bots -= 1
self.send(client, '5' + json.dumps([client.id]))
threading.Thread(target=self.startGame, args=(), daemon=True).start()
self.sendingImages(client)
self.sendAllImages(client)
```

14.2. Every player starts with the same amount of starting chips ten white chips, five green, three red, two blue, and one black chip. (white = 1, green = 2, red = 4, blue = 8, black = 16)

```
self.money = [10, 5, 3, 2, 1]
```

14.3. At the start of each round, the player and bots are dealt two unique cards from the deck.

```
for client in self.clients:
    client.hand = [self.deck.getCard(), self.deck.getCard()]
```

14.4. Assign a dealer at the start of each round and make the player clockwise to the dealer the little blind and the next player the big blind.

14.4.1. Make the little blind bet the minimum bet.

14.4.2. Make the big blind bet double the little blind.

```
self.clients[(self.playerNumber) % len(self.clients)].setRole('Dealer')
self.clients[(self.playerNumber + 1) % len(self.clients)].setRole('Little
Blind')
self.clients[(self.playerNumber + 1) % len(self.clients)].betting = 1
self.clients[(self.playerNumber + 1) % len(self.clients)].lastbet = 1
self.lastbet = 1
betAmount = self.clients[(self.playerNumber + 1) %
len(self.clients)].bet(self.lastbet)
self.addChipsToPot(betAmount)
self.clients[(self.playerNumber + 2) % len(self.clients)].setRole('Big
Blind')
self.clients[(self.playerNumber + 2) % len(self.clients)].betting = 2
betAmount = self.clients[(self.playerNumber + 2) %
len(self.clients)].bet(self.lastbet)
self.clients[(self.playerNumber + 2) % len(self.clients)].lastbet = 2
self.lastbet = 2
self.addChipsToPot(betAmount)
```

14.5. The player which would play first in a round would be the player next to the big blind.

```
self.playerNumber = (self.playerNumber + 3) % len(self.clients)
```

14.6. Check if the current player has folded, if so, skip their go.

```
if not self.currentPlayer.folded:
```

14.6.1. If the player hasn't folded prompt them to make their go.

14.6.1.1. Players and bots should be able to fold, and bet.

```
self.currentPlayer.promptAction()
```

14.7. If everyone has bet the same the first three community card are then revealed

14.7.1. The players are then looped over again, if everyone bet the same then the 4th community card is revealed.

14.7.2. The players are then looped over again, if everyone bet the same then the 5th and final card is revealed.

```
self.showCommunityCards()
```

14.7.3. The showdown is then played until everyone bets the same.

14.7.3.1. The player with the highest value hand wins the pot or is split among the joint highest hands.

```
winners = self.returnWinners()
self.rewardPlayers(winners)
self.pot = [0, 0, 0, 0, 0]
```

14.8. If every player but one has folded, then the last player wins the pot regardless.

```
for player in self.clients:
            if not player.folded:
                player.handValue = compare.getValueOfHand(player.hand
+self.communityCards)
            else:
                player.handValue = 0
        maxHandValue = max(self.clients, key=lambda
c:c.handValue).handValue
        print('best hand ever', maxHandValue)
        winners = list(filter(lambda c: c.handValue >= maxHandValue and
not c.folded, self.clients))
        print('these people have won', winners)
        return winners
```

14.9. After the winner has received their winnings, a short intermission will occur, afterward a new round will be played.

```
time.sleep(5)
self.startOfNewRound()
```

14.10. If the player quits, by either closing the game or returning to the main menu, the game will end.

15. Multiplayer

15.1. Similar to single player but allows other players to be able to join and play in the same game.

15.2. Empty player slots are filled with bots.

15.2.1. If a player joins mid game, the game will wait for the round to end before removing a bot and adding the player to the game.

15.3. If the game is full of players, then the user trying to connect to server will not be connected to server.

```
connection, address = self.server.accept()
                    connection.settimeout(1)
                    playerInfo = self.recieveData(connection)
```

```
                    playerImage = self.recievingImages(connection)
                    print('The bluetooth device is a connected-
uhsuccessfulay to:', address)
                    if not self.gameStarted:
                        client = ServerClient(connection, address[0],
username=playerInfo['Name'], pfp=playerImage)
                        self.clients[self.players] = client
                        self.players += 1
                        self.bots -= 1
                        self.send(client, '5' + json.dumps([client.id]))
                        if client.isHost():

threading.Thread(target=self.startGame,args=(), daemon=True).start()

                        self.sendingImages(client)
                        self.sendAllImages(client)
                    elif len(self.waitingRoom) <= self.bots:
                        self.waitingRoom.append(ServerClient(connection,
address[0], username=playerInfo['Name'], pfp=playerImage))
                    else:
                        connection.close()
```

15.4. Every player starts with the same amount of starting chips ten white chips, five green, three red, two blue, and one black chip.

```
self.money = [10, 5, 3, 2, 1]
```

15.5. At the start of each round, the player and bots are dealt two unique cards from the deck.

```
for client in self.clients:
   client.hand = [self.deck.getCard(), self.deck.getCard()]
```

15.6. Assign a dealer at the start of each round and make the player clockwise to the dealer the little blind and the next player the big blind.

15.6.1. Make the little blind bet the minimum bet.

15.6.2. Make the big blind bet double the little blind.

```
self.clients[(self.playerNumber) % len(self.clients)].setRole('Dealer')
self.clients[(self.playerNumber + 1) %
len(self.clients)].setRole('Little Blind')
self.clients[(self.playerNumber + 1) % len(self.clients)].betting = 1
self.clients[(self.playerNumber + 1) % len(self.clients)].lastbet = 1
self.lastbet = 1
betAmount = self.clients[(self.playerNumber + 1) %
len(self.clients)].bet(self.lastbet)
self.addChipsToPot(betAmount)
self.clients[(self.playerNumber + 2) % len(self.clients)].setRole('Big
Blind')
self.clients[(self.playerNumber + 2) % len(self.clients)].betting = 2
betAmount = self.clients[(self.playerNumber + 2) %
len(self.clients)].bet(self.lastbet)
self.clients[(self.playerNumber + 2) % len(self.clients)].lastbet = 2
self.lastbet = 2
self.addChipsToPot(betAmount)
```

15.7. The player which would play first in a round would be the player next to the big blind.

```
self.playerNumber = (self.playerNumber + 3) % len(self.clients)
```

15.8. Check if the current player has folded, if so, skip their go.

```
if not self.currentPlayer.folded:
```

15.8.1. If the player hasn't folded prompt them to make their go.

15.8.1.1. Players and bots should be able to fold, and bet.

```
self.currentPlayer.promptAction()
```

15.9. If everyone has bet the same the first three community card are then revealed

15.9.1. The players are then looped over again, if everyone bet the same then the 4th community card is revealed.

15.9.2. The players are then looped over again, if everyone bet the same then the 5th and final card is revealed.

```
self.showCommunityCards()
```

15.9.3. The showdown is then played until everyone bets the same.

15.9.3.1. The player with the highest value hand wins the pot or is split among the joint highest hands.

```
winners = self.returnWinners()
self.rewardPlayers(winners)
self.pot = [0, 0, 0, 0, 0]
```

15.10. If every player but one has folded, then the last player wins the pot regardless.

```
for player in self.clients:
            if not player.folded:
                player.handValue = compare.getValueOfHand(player.hand
+self.communityCards)
            else:
                player.handValue = 0
        maxHandValue = max(self.clients, key=lambda
c:c.handValue).handValue
        print('best hand ever', maxHandValue)
        winners = list(filter(lambda c: c.handValue >= maxHandValue and
not c.folded, self.clients))
        print('these people have won', winners)
        return winners
```

15.11. After the winner has received their winnings, a short intermission will occur, afterward a new round will be played.

```
time.sleep(5)
self.startOfNewRound()
```

15.12. If a player quits, by either closing the game or returning to the main menu, the game will end for the user that left.

15.12.1. If they are the host, the server will close.

```
client.connected = False
                            self.players -= 1
                            if client in self.waitingRoom:
                                self.waitingRoom.remove(client)
                            else:
                                botName = getBotName()
```

```
                                    bot = ServerBot(username='Bot ' +
botName, pfp=botImages[botName], server=self)
                                    if self.gameStarted:
                                        client.folded = True
                                        client.lastAction = 'Folded'
                                        bot.fold()
                                        bot.id = client.id
                                        bot.hand = [Card(suit='Spades',
value=1)] * 2
                                        # bot.money = [10, 5, 3, 2, 1]
                                        bot.lastAction = 'Sat Out'
                                    self.clients[self.clients.index(client)]
= bot
                                    if self.gameStarted and
self.currentPlayer and self.currentPlayer.id == client.id:
                                        self.endGo()
                                client.connection.close()
                                if self.players <= 0:
                                    self.close()
```

15.13. The players in the game will be able to communicate with one another via an in-game chat.

```
for c in self.clients:
                                    if not c.isBot(): # client.id != c.id
and
                                        c.addToQueue(b'7' + bytes(data,
'utf-8') + b'\x1e')
```

16. In-Game Character

16.1. Allow the user, under the settings menu, to be able to change their name and profile picture.

16.1.1. Placing an image in the 'User Images' folder and then specifying the name of the image in game under the settings.

16.1.1.1. If no image is selected or is unable to be loaded, a default one is chosen.

16.1.2. Allow users to change their in-game name (maximum of 30 characters)

```
def settingsMenu(self):
        self.removeAllSprites()
        self.background = self.grey
        self.userText = TextLabel(self, text='Username:',
xy=[self.width/2 - 180, 150], group=self.objects)
        self.pictureText = TextLabel(self, text='Picture:',
xy=[self.width/2 - 160, 186], group=self.objects)
        self.portText = TextLabel(self, text='Port:', xy=[self.width/2 -
140, 222], group=self.objects)
        self.widthText = TextLabel(self, text='Width:', xy=[self.width/2
- 150, 318], group=self.objects)
        self.heightText = TextLabel(self, text='Height:',
xy=[self.width/2 - 150, 354], group=self.objects)
        self.t1 = TextInput(self, default=settings['Username'],
xy=[self.width/2 - 100, 150], function=self.writeSettings,
param='Username', group=self.objects)
        self.t2 = TextInput(self, default=settings['Picture'],
xy=[self.width/2 - 100, 186], function=self.writeSettings,
param='Picture', group=self.objects)
```

```
        self.t3 = TextInput(self, default=settings['Server Port'],
xy=[self.width/2 - 100, 222], function=self.writeSettings, param='Server
Port', group=self.objects)
        self.b1 = Button(self, xy=[self.width/2, 287],
function=self.updateFullscreen, text='Fullscreen Toggle' , dim=[250,
50], group=self.objects)
        self.fScreentxt = TextLabel(self, text='(restart to take
effect)', xy=[self.width/2 + 290, 272], group=self.objects)
        self.t4 = TextInput(self, default=settings['Width'],
xy=[self.width/2 - 100, 318], function=self.writeSettings,
param='Width', group=self.objects)
        self.t5 = TextInput(self, default=settings['Height'],
xy=[self.width/2 - 100, 354], function=self.writeSettings,
param='Height', group=self.objects)
        self.b2 = Button(self, xy=[self.width/2, 419],
function=self.mainMenu, text='Back', dim=[200, 50], group=self.objects)
```

16.2. During gameplay allow the user to increase and decrease their bet, check and fold.

```
if self.parent.parent.client.turn:
            if keys[pygame.K_UP]:
                self.parent.parent.client.increaseBet()
            elif keys[pygame.K_DOWN]:
                self.parent.parent.client.decreaseBet()
            elif keys[pygame.K_f]:
                self.parent.parent.client.betting = 0
                self.parent.parent.client.fold()
```

16.3. Allow the user to peek at their hand when holding down the spacebar.

```
if keys[pygame.K_SPACE]: self.flipHand()
```

16.4. In game the user's chips should be drawn on the screen so that the player knows their chips

16.5. The users profile picture and username should be drawn on the screen so that they know what they look like to other players.

```
self.mainPlayerHUD = MainPlayerUI(self, size=100,
picture=self.parent.client.pfp, username=self.parent.client.username)
```

Testing

To make sure that my program is working to specification, it is crucial that I test my software to make sure that there are no bugs which could overall affect my user's main experience.

So that you can see what I tested and how I did so, below is a YouTube video link as well as the table for my tests.

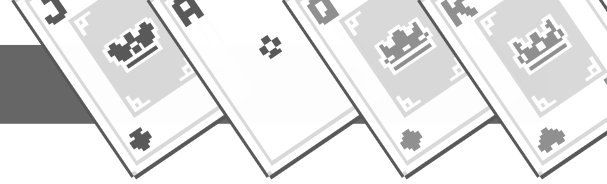
QR Code to testing video:

Testing video link (for redundancy):

Menu Testing

Test #	Objective #	Test/Input Data	Purpose/Reasoning	Expected Result	Actual Result	Video Timestamp
1.0	1.1	Running the game.	To see if the main menu will load for the player	The game loads the main menu	The game loads the main menu.	
1.1	1.2 + 1.2.1	Pressing the 'Play' button on the main screen.	To see if as a user you can proceed to the next menu where you are able to select game type.	The game progresses to the next menu.	Immediately after being clicked the game shows the game modes menu.	
1.2	1.2.1.2.3	Pressing the 'Back' button on the game selection menu.	To see if you can go back to the main menu	The game changes menu back to the main menu.	The game changes the menu back to the main menu.	
1.3	1.2.1.1	Pressing the 'Singleplayer' button on the game selection menu.	To see if a user can start a single player game	The game starts a single player game.	When clicking the button the game screen changes, after a short delay the game starts.	
1.4	1.2.1.2	Pressing the 'Multiplayer' button on the game selection menu.	To see if a user can proceed to the join or create server menu.	The game changes the menu to the join or create server menu.	The game shows the join button, create button, and back button.	
1.5	1.2.1.2.3	Pressing the back button on the join or creation menu.	To see if a user can go backward to the game selection menu.	The game changes the menu back towards the game selection menu	The game changes the menu back towards the game selection menu	
1.6	1.2.1.2.1	Pressing the join server button on join or create menu.	To see if a user can get to the screen where they can enter the server details.	The game changes the menu over to the menu allowing the user to input data.	The game changes the screen with the text boxes allowing the user to input data.	

				server details.		
1.7	1.2.1.2.3	Pressing the 'Back' button on the join server screen	To see if a user can go back towards the previous menu	The previous menu is shown to the player	The game shows the previous menu where they can choose to join, create, or go back.	
1.8	1.2.1.2.2	Pressing the 'create lobby' button on the join or create menu	To see if a user can start a multiplayer server	The game creates a multiplayer server.	When clicking the button the game screen changes, after a short delay the game starts.	
1.9	1.3	Pressing the settings button in the main menu.	To see if the player can access the settings menu.	The game will update the screen to the settings menu.	The game loads the settings menu where the player can type in text boxes and press buttons.	
1.10	1.2.1.2.3	Pressing the back button in the settings menu.	The user can access the main menu.	The game will update the game back to the main menu.	The game loads the main menu.	
1.11	1.3.1.1 + 1.3.1.2 + 9	Updating settings in game	To see if the user can input data in the settings and update them.	The game will update all the settings apart from screen resolution and size.	The game updates the username and profile picture shown in game to the user chosen ones	
1.12	1.3.1.2	Entering text into a text box.	To check if the character limits work.	The game stops the user from entering any more text if it exceeds 30 characters.	The game stops the user from being able to write strings longer than 30 characters.	
1.13	1.4	Pressing the quit button in the main menu.	To see if the user can close the game.	The game will close.	After a second the game will close.	



Singleplayer Testing

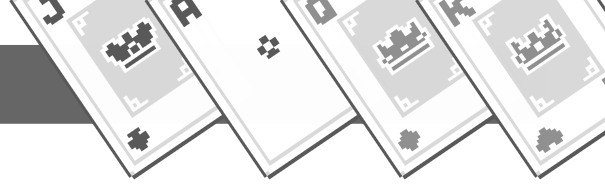
Test #	Objective #	Test/Input Data	Purpose/Reasoning	Expected Result	Actual Result	Video Timestamp
2.0	2.1	Starting the game.	To see if the game will start with four other bots.	The game is made and then starts with four other bots.	The game is made and then starts with four other bots.	
2.1	2.2	Playing singleplayer.	To see if the game will give all bot and the player the correct starting amount.	The game makes all players start with 10 white chips, 5 green, 3 red, 2 blue, and one black.	The game makes all players start with 10 white chips, 5 green, 3 red, 2 blue, and one black.	
2.2	2.3	Playing singleplayer.	To see if the game assigns two random cards from the deck.	The game should assign the player and the bot with 2 different cards.	The game assigns the player 2 different cards from the deck.	
2.3	2.4	Playing singleplayer.	To see if the correct roles on the players are assigned.	The game assigns a dealer, big blind, and little blind.	The game assigns 3 players with roles, dealer, little blind, and blind.	
2.4	2.4.1	Playing singleplayer	To see if the little bind is made to bet the minimum	The game makes the player which is the little blind bet 1 at the start.	When the game starts the player who is the little blind is made to bet 1	
2.5	2.4.2	Playing singleplayer	To see if the big blind is made to bet double the little blind	The game makes the player which is the big blind bet 2 at the start.	When the game starts the player who is the big blind is made to bet 2	

2.6	2.5	Playing singleplayer	To see if the player who turn it is after the big blind.	The person next to the little blind is first to play.	The person next to the little blind is prompted to play first.	
2.7	2.6	Making a bet.	To make sure that the game prompts players correctly and ignores folded players,	The game correctly prompts non folded players and allows them to play.	Folded players are ignored, and playing players are prompted to play.	
2.8	2.7	Playing singleplayer	To make sure that the flop is shown when the bet is the same,	When every player bet the same, three community cards are revealed.	When everyone bets equal amounts, the community cards are revealed at the top of the screen.	
2.9	2.7.1	Playing singleplayer	To make sure that the turn is shown after the flop.	When every player bet the same, one additional card is revealed.	When everyone bets equal amounts, the new community cards are revealed at the top of the screen.	
2.10	2.7.2	Playing singleplayer	To make sure that the river is shown after turn.	When every player bet the same, one additional card is revealed.	When everyone bets equal amounts, the new community cards are revealed at the top of the screen.	
2.11	2.7.3	Playing singleplayer	To make sure players can bet after until everyone's bet is the same. Then the winner is shown.	When every player bet the same, the game end and the winning player is credited	The get winning players function fails to work and causes an error.	
2.12	2.8	Playing singleplayer	To make sure that if one player doesn't fold and the rest have, that they win.	If one player remains, then the game will credit that player with	The last remaining player wins the money in the pot regardless of their hand.	

				the pot money.		
2.13	2.9	Playing singleplayer	To make sure that the game progresses correctly.	The game starts a new round with reset statistics.	The game starts again, with a new hand and reset pot.	
2.14	2.4	Playing singleplayer	To check if the correct roles are assigned when a new game is made	The dealer is incremented by one, there foremoving the small blind and big blind.	The game does not increment the dealer.	
2.15	2.10	Pressing the quit button in the pause menu.	To check if the game will quit if the player decides to leave.	The game quits when the quit button is activated.	The game quits when the quit button is activated.	
2.16	2.10	Pressing the main menu button in the pause menu.	To check if game will allow the player back to the main menu.	The game sends the user back to the main menu.	The game sends the user back to the main menu.	

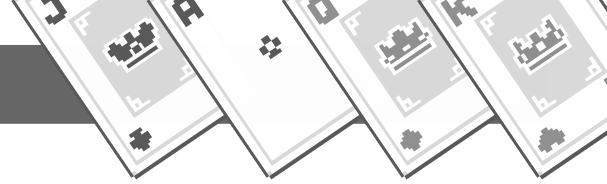
Multiplayer Testing

Test #	Objective #	Test/Input Data	Purpose/Reasoning	Expected Result	Actual Result	Video Timestamp
3.0	3.1 + 3.2	Starting the game.	To see if the game allows players in and fills empty slots with bots	The game is made, and players are shown on the screen with bots filling the empty slots.	The game is made and then starts with the amount of players connected and the rest of the slots are filled with bots.	
3.1	3.4	Playing multiplayer.	To see if the game will give all bot and the player the correct starting amount.	The game makes all players start with 10 white chips, 5 green, 3 red, 2 blue, and one black.	The game makes all players start with 10 white chips, 5 green, 3 red, 2 blue, and one black.	
3.2	3.5	Playing multiplayer.	To see if the game assigns two random cards from the deck.	The game should assign the player and the bot with 2 different cards.	The game assigns the player 2 different cards from the deck.	
3.3	3.6	Playing multiplayer.	To see if the correct roles on the players are assigned.	The game assigns a dealer, big blind, and little blind.	The game assigns 3 players with roles, dealer, little blind, and blind.	
3.4	3.6.1	Playing multiplayer.	To see if the little bind is made to bet the minimum	The game makes the player which is the little blind bet 1 at the start.	When the game starts the player who is the little blind is made to bet 1	
3.5	3.6.2	Playing multiplayer.	To see if the big blind is made to bet double the little blind	The game makes the player which is the big blind bet	When the game starts the player who is the big blind is made to bet 2	



				2 at the start.		
3.6	3.7	Playing multiplayer.	To see if the player who turn it is after the big blind.	The person next to the little blind is first to play.	The person next to the little blind is prompted to play first.	
3.7	3.8	Making a bet.	To make sure that the game prompts players correctly and ignores folded players,	The game correctly prompts non folded players and allows them to play.	Folded players are ignored, and playing players are prompted to play.	
3.8	3.9	Playing multiplayer.	To make sure that the flop is shown when the bet is the same,	When every player bet the same, three community cards are revealed.	When everyone bets equal amounts, the community cards are revealed at the top of the screen.	
3.9	3.9.1	Playing multiplayer.	To make sure that the turn is shown after the flop.	When every player bet the same, one additional card is revealed.	When everyone bets equal amounts, the new community cards are revealed at the top of the screen.	
3.10	3.9.2	Playing multiplayer.	To make sure that the river is shown after turn.	When every player bet the same, one additional card is revealed.	When everyone bets equal amounts, the new community cards are revealed at the top of the screen.	
3.11	3.9.3	Playing multiplayer.	To make sure players can bet after until everyone's bet is the same. Then the winner is shown.	When every player bet the same, the game end and the winning player is credited	The get winning players function fails to work and causes an error.	
3.12	3.10	Playing multiplayer.	To make sure that if one player doesn't fold and the	If one player remains, then the	The last remaining player wins the	

			rest have, that they win.	game will credit that player with the pot money.	money in the pot regardless of their hand.	
3.13	3.11	Playing multiplayer.	To make sure that the game progresses correctly.	The game starts a new round with reset statistics.	The game starts again, with a new hand and reset pot.	
3.14	3.6	Playing singleplayer	To check if the correct roles are assigned when a new game is made	The dealer is incremented by one, there foremoving the small blind and big blind.	The game does not increment the dealer.	
3.15	3.13	Pressing the quit button in the pause menu.	To check if the game will quit if the player decides to leave.	The game quits when the quit button is activated.	The game quits when the quit button is activated.	
3.16	3.13	Pressing the main menu button in the pause menu.	To check if game will allow the player back to the main menu.	The game sends the user back to the main menu.	The game sends the user back to the main menu.	
3.17	3.13	Letting another player quit.	To check if they get removed, and then replaced by a bot later.	The player is folded and then replaced by a bot.	The player which left the game is folded and then replaced by a bot.	
3.18	3.2.1	Player joins mid match.	To check if the player is put into a waiting list and is then able to play next round.	The game puts the player in waiting, and when the game has finished, adds them in.	The game puts the player in waiting, and when the game has finished, adds them in on the screen for the player to see.	
3.19	3.13 + 11.1.4	Sending and receiving messages.	To check if players can communicate with each other.	The game allows players to send message and receive them in the chat.	The game doesn't receive the messages correctly	



3.20	11.1.5	Pressing the alt key	Check if the chat can be closed and opened.	The chat if open closes, and otherwise opens.	The chat can be operated between an open and closed state.	

UI Testing

Test #	Objective #	Test/Input Data	Purpose/Reasoning	Expected Result	Actual Result	Video Timestamp
4.0	5.1	Loading singleplayer or multiplayer.	Checking if the community cards load and display properly.	Community cards are displayed on the top of the screen.	The currently shown community cards are displayed.	
4.1	5.2 + 5.2.1 + 5.2.2.3	Loading singleplayer or multiplayer.	Check if the players are displayed correctly.	The current players partaking in a match are shown along the sides of the screen.	Players partaking in the match are shown around the edge along with their name and profile picture, status and chips.	
4.2	5.2.2.1 + 5.2.2.2	Loading singleplayer or multiplayer.	Check if the last action of a player is shown correctly.	Text is displayed next to the user with the action they took.	Fold, call, or raise is shown if a player does any of these actions.	
4.3	5.3	Loading singleplayer or multiplayer.	See if the controls are drawn properly.	Controls along with text next to the button is displayed in the bottom left.	Controls along with text next to the button is displayed.	
4.4	5.4	Loading singleplayer or multiplayer.	See if the game details are correctly displayed.	The game time and current player should be displayed	The game time and current player is displayed at the top middle above the	

				on the screen.	community cards.	
4.5	8.1	Pressing tab in game.	To see if the player is able to access poker help in game.	The game displays information to the player which helps them if they are unsure.	The GUI is rendered below the other UI elements.	

Testing Fixes

Test 2.11

```
def returnWinners(self):
        for player in self.clients:
            if not player.folded:
                player.handValue = compare.getValueOfHand(player.hand +
self.communityCards)
            else:
                player.handValue = 0
        maxHandValue = max(self.clients, key=lambda c:
c.handValue).handValue
        print('best hand ever', maxHandValue)
        winners = list(filter(lambda c: c.handValue >= maxHandValue and not
c.folded, self.clients))
        print('these people have won', winners)
        return winners
```

Test 2.14

```
def startOfNewRound(self):

        self.gameNumber += 1
        self.playerNumber = self.currentDealer
        self.lastbet = 0
        self.communityCards = []

        for idx, client in enumerate(self.clients):
            if client.isBot() and len(self.waitingRoom) > 0:
                self.clients[idx] = self.waitingRoom.pop(0)

        for client in self.clients:
            client.setPlayingStatus(True)
            client.setRole('Player')
            client.lastbet = 0
            client.handValue = 0
            client.folded = False
            client.hand = []
            client.lastAction = ''
            client.betting = 0
            client.turn = False
            if not client.isBot():
                client.isLoaded = False
        self.clients[(self.playerNumber) %
len(self.clients)].setRole('Dealer')
        self.clients[(self.playerNumber + 1) %
len(self.clients)].setRole('Little Blind')
        self.clients[(self.playerNumber + 1) % len(self.clients)].betting =
1
        self.clients[(self.playerNumber + 1) % len(self.clients)].lastbet =
1
        self.lastbet = 1
        betAmount = self.clients[(self.playerNumber + 1) %
len(self.clients)].bet(self.lastbet)
        self.addChipsToPot(betAmount)
        self.clients[(self.playerNumber + 2) %
len(self.clients)].setRole('Big Blind')
        self.clients[(self.playerNumber + 2) % len(self.clients)].betting =
2
        betAmount = self.clients[(self.playerNumber + 2) %
len(self.clients)].bet(self.lastbet)
```

```
        self.clients[(self.playerNumber + 2) % len(self.clients)].lastbet =
2
        self.lastbet = 2
        self.addChipsToPot(betAmount)
        whoBet = (self.playerNumber + 2) % len(self.clients)#
        basebetNumber = self.lastbet

        self.playerNumber = (self.playerNumber + 3) % len(self.clients)

        self.deck.regenDeck()

        for client in self.clients:
            client.hand = [self.deck.getCard(), self.deck.getCard()]

        self.updatePlayers()

        self.sendAllClients("3[]")

        self.waitForPlayers()
        self.sendAllClients('8' + json.dumps({
            'Start Time' : time.time()
        }))
        # pre flop, flop, the turn, the river, the showdown
        endGame = False
        for i in range(5):
            if endGame:
                break
            for i in range(len(self.clients)):
                print('Next Go')
                self.currentPlayer = self.clients[self.playerNumber]
                self.updatePlayers()
                if not self.currentPlayer.folded:
                    self.timer = threading.Timer(self.turnWait, self.endGo,
args=(True,))
                    self.timer.start()
                    self.currentPlayer.promptAction()
                    self.timer.join()

                print('testing data:', self.currentPlayer.lastbet,
basebetNumber, self.currentPlayer.folded, whoBet)
                self.playerNumber = (self.playerNumber + 1) %
len(self.clients)
                foldCount = 0
                for player in self.clients:
                    if player.folded:
                        foldCount += 1
            for player in self.clients:
                player.lastbet = 0
            self.lastbet = 0
            print('mate')
            self.showCommunityCards()
        winners = self.returnWinners()
        self.rewardPlayers(winners)
        self.pot = [0, 0, 0, 0, 0]
        time.sleep(5)
        self.currentDealer += 1
        self.startOfNewRound()
```

Test 3.11

```
def returnWinners(self):
        for player in self.clients:
```

```
            if not player.folded:
                player.handValue = compare.getValueOfHand(player.hand +
self.communityCards)
            else:
                player.handValue = 0
        maxHandValue = max(self.clients, key=lambda c:
c.handValue).handValue
        print('best hand ever', maxHandValue)
        winners = list(filter(lambda c: c.handValue >= maxHandValue and not
c.folded, self.clients))
        print('these people have won', winners)
        return winners
```

Test 3.14

```
def startOfNewRound(self):

        self.gameNumber += 1
        self.playerNumber = self.currentDealer
        self.lastbet = 0
        self.communityCards = []

        for idx, client in enumerate(self.clients):
            if client.isBot() and len(self.waitingRoom) > 0:
                self.clients[idx] = self.waitingRoom.pop(0)

        for client in self.clients:
            client.setPlayingStatus(True)
            client.setRole('Player')
            client.lastbet = 0
            client.handValue = 0
            client.folded = False
            client.hand = []
            client.lastAction = ''
            client.betting = 0
            client.turn = False
            if not client.isBot():
                client.isLoaded = False
        self.clients[(self.playerNumber) %
len(self.clients)].setRole('Dealer')
        self.clients[(self.playerNumber + 1) %
len(self.clients)].setRole('Little Blind')
        self.clients[(self.playerNumber + 1) % len(self.clients)].betting =
1
        self.clients[(self.playerNumber + 1) % len(self.clients)].lastbet =
1
        self.lastbet = 1
        betAmount = self.clients[(self.playerNumber + 1) %
len(self.clients)].bet(self.lastbet)
        self.addChipsToPot(betAmount)
        self.clients[(self.playerNumber + 2) %
len(self.clients)].setRole('Big Blind')
        self.clients[(self.playerNumber + 2) % len(self.clients)].betting =
2
        betAmount = self.clients[(self.playerNumber + 2) %
len(self.clients)].bet(self.lastbet)
        self.clients[(self.playerNumber + 2) % len(self.clients)].lastbet =
2
        self.lastbet = 2
        self.addChipsToPot(betAmount)
        whoBet = (self.playerNumber + 2) % len(self.clients)#
        basebetNumber = self.lastbet
```

```
        self.playerNumber = (self.playerNumber + 3) % len(self.clients)

        self.deck.regenDeck()

        for client in self.clients:
            client.hand = [self.deck.getCard(), self.deck.getCard()]

        self.updatePlayers()

        self.sendAllClients("3[]")

        self.waitForPlayers()
        self.sendAllClients('8' + json.dumps({
            'Start Time' : time.time()
        }))
        # pre flop, flop, the turn, the river, the showdown
        endGame = False
        for i in range(5):
            if endGame:
                break
            for i in range(len(self.clients)):
                print('Next Go')
                self.currentPlayer = self.clients[self.playerNumber]
                self.updatePlayers()
                if not self.currentPlayer.folded:
                    self.timer = threading.Timer(self.turnWait, self.endGo,
args=(True,))
                    self.timer.start()
                    self.currentPlayer.promptAction()
                    self.timer.join()

                print('testing data:', self.currentPlayer.lastbet,
basebetNumber, self.currentPlayer.folded, whoBet)
                self.playerNumber = (self.playerNumber + 1) %
len(self.clients)
                foldCount = 0
                for player in self.clients:
                    if player.folded:
                        foldCount += 1
            for player in self.clients:
                player.lastbet = 0
            self.lastbet = 0
            print('mate')
            self.showCommunityCards()
        winners = self.returnWinners()
        self.rewardPlayers(winners)
        self.pot = [0, 0, 0, 0, 0]
        time.sleep(5)
        self.currentDealer += 1
        self.startOfNewRound()
```

Evaluation

Objectives

How have the objectives have been met?

1. Main Menu
    a. The user of the program upon opening the game, should be greeted with the main options of the game which include a play button, a settings button, and a quit button.

    When the player opens the game they are shown the options to change settings and play or quit the game.

    b. User of the program should be able to select the play button.
        i. Upon clicking this button, the menu will be updated to show new buttons relating to the button the user just pressed, this being the play button. Users will be asked using buttons whether they would like to play single player or multiplayer or return to the main menu.
        ii. Clicking the single player button should start a single player session with 4 other bots who will play along with the player until the user decides to quit.
        iii. Clicking the multiplayer button will send the user to a new menu with buttons asking if the user would like to join a server or create one as well as return to the previous menu.

        When the user presses the play button they are able to choose whether they would like to play a singleplayer game, a multiplayer game or return to the previous menu.

            1. Clicking the join lobby button will send the user to another menu where they will be prompted to enter the server ip as well as the server port and then be able to click 'Join' to join that server.
            2. Clicking create server will create a server hosting a game for the user and connect that user, then allowing other users to enter the game.
            3. Previous button should return the user back to the previous menu.

            If the user selects singleplayer, a single player game will be created. However, if it is multiplayer, there is another menu prompting the player if they would like to join a lobby or create one, as well as the option to return to the previous menu.

    c. User of the program should be able to select the settings button.
        i. User will be shown settings that they can change such as username, default port on which servers will be made, full screen, and screen size.

ii. Toggleable settings are a button.
iii. Text boxes, where written inputs are entered, pressing enter will save the information.

The player once selecting the settings menu is able to change aspects of the game such as their profile picture, their name, the port used to create lobbies and their screen dimensions, these fields only are saved when enter is pressed.

d. User of the program should be able to select the quit button.
    i. Upon clicking select the game should close

    If the player presses the quit button in game, it will close their game down.

2. Single player
    a. If a single player game is made, then the game will fill the other four player slots with bots so that the player is able to play against opponents.

    Singleplayer games start with one player and four randomly chosen bots for the user to play against.

    b. Every player starts with the same amount of starting chips ten white chips, five green, three red, two blue, and one black chip. (white = 1, green = 2, red = 4, blue = 8, black = 16)

    The player and all bots start with the correct set amount of chips, 10, 5, 3, 2, 1

    c. At the start of each round, the player and bots are dealt two unique cards from the deck.

    The player is given two random cards at the start of each round, including bots.

    d. Assign a dealer at the start of each round and make the player clockwise to the dealer the little blind and the next player the big blind.
        i. Make the little blind bet the minimum bet.
        ii. Make the big blind bet double the little blind.

        The game forces the first two minimum bets at the start of every round.

    e. The player which would play first in a round would be the player next to the big blind.

    The first player to start is the player next to the big blind.

f. Check if the current player has folded, if so, skip their go.
    i. If the player hasn't folded prompt them to make their go.
    ii. Players and bots should be able to fold, and bet.

    The player and bots, if it is their go are prompted to make their go, unless they are folded then they are skipped. The player and bots can choose to raise, call, check, and call.

g. If everyone has bet the same the first three community card are then revealed
    i. The players are then looped over again, if everyone bet the same then the 4th community card is revealed.
    ii. The players are then looped over again, if everyone bet the same then the 5th and final card is revealed.
    iii. The showdown is then played until everyone bets the same.
    iv. The player with the highest value hand wins the pot or is split among the joint highest hands.

    The game check to see if the bet in the round hasn't increased, and if it has, it waits until everyone bets the same before showing the community cards.
    The cards are shown in stages called the flop, the river, and the turn. Finally ending on the showdown where the player with best hand wins, where as if there are more than one winner, the winnings are split.

h. If every player but one has folded, then the last player wins the pot regardless.

    If all but one player folds, the last remaining player wins the entire pot.

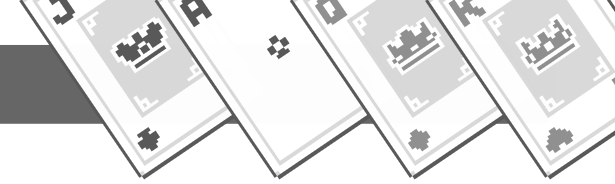
i. After the winner has received their winnings, a short intermission will occur, afterward a new round will be played.

    When the winner(s) have received their winnings, there is a short five second intermission.

j. If the player quits, by either closing the game or returning to the main menu, the game will end.

    When the player quits, the game stops and closes.

3. Multiplayer
    a. Similar to single player but allows other players to be able to join and play in the same game.
    b. Empty player slots are filled with bots.



i. If a player joins mid game, the game will wait for the round to end before removing a bot and adding the player to the game.

The game will fill empty slots with bots, and if a player joins mid game, they will be added to a waiting list and added at the start of a new round.

c. If the game is full of players, then the user trying to connect to server will not be connected to server.

The user attempting to connect to a server which is fill is sent back to the main menu.

d. Every player starts with the same amount of starting chips ten white chips, five green, three red, two blue, and one black chip.

Every player and bot start with the set amount of chips, 10, 5, 3, 2, 1

e. At the start of each round, the player and bots are dealt two unique cards from the deck.

Every player and bot are given two random cards dealt from the in-game deck.

f. Assign a dealer at the start of each round and make the player clockwise to the dealer the little blind and the next player the big blind.
   i. Make the little blind bet the minimum bet.
   ii. Make the big blind bet double the little blind.

The big blind and little blind are forced to take the minimum bet at the start of every new round.

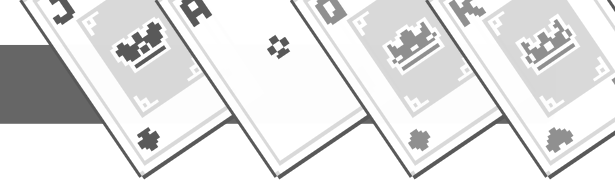
g. The player which would play first in a round would be the player next to the big blind.

The first player to play in a new game is the player next to the big blind.

h. Check if the current player has folded, if so, skip their go.
   i. If the player hasn't folded prompt them to make their go.
   ii. Players and bots should be able to fold, and bet.

The player and bots, if it is their go are prompted to make their go, unless they are folded then they are skipped. The player and bots can choose to raise, call, check, and call.

i. If everyone has bet the same the first three community card are then revealed
   i. The players are then looped over again, if everyone bet the same then the 4th community card is revealed.



ii. The players are then looped over again, if everyone bet the same then the 5th and final card is revealed.
iii. The showdown is then played until everyone bets the same.
iv. The player with the highest value hand wins the pot or is split among the joint highest hands.

The game check to see if the bet in the round hasn't increased, and if it has, it waits until everyone bets the same before showing the community cards.
The cards are shown in stages called the flop, the river, and the turn. Finally ending on the showdown where the player with best hand wins, where as if there are more than one winner, the winnings are split.

j. If every player but one has folded, then the last player wins the pot regardless.

If all but one player folds, the last remaining player wins the entire pot.

k. After the winner has received their winnings, a short intermission will occur, afterward a new round will be played.

When the winner(s) have received their winnings, there is a short five second intermission.

l. If a player quits, by either closing the game or returning to the main menu, the game will end for the user that left.
    i. If they are the host, the server will close.

When a player quits, the game will show them as 'sat out' to other players before replacing them with a bot at the start of the new round, Whereas if they are the host the server will close, and the connected players are disconnected.

m. The players in the game will be able to communicate with one another via an in-game chat.

Users are able to communicate via text chat in game.

4. In-Game Character
    a. Allow the user, under the settings menu, to be able to change their name and profile picture.
        i. Placing an image in the 'User Images' folder and then specifying the name of the image in game under the settings.
        ii. If no image is selected or is unable to be loaded, a default one is chosen.

iii. Allow users to change their in-game name (maximum of 30 characters)

The user is able to change their user name in a text box, and select a profile picture which they have placed in the user images folder.

b. During gameplay allow the user to increase and decrease their bet, check and fold.

The player is able to adjust their bet, and fold in game.

c. Allow the user to peek at their hand when holding down the spacebar.

When the space bar is pressed, the hand is shown to the player as well as the value of the hand above.

d. In game the user's chips should be drawn on the screen so that the player knows their chips

The players chips are drawn in the middle bottom of the screen directly in front of the player, making it easy to see.

e. The users profile picture and username should be drawn on the screen so that they know what they look like to other players.

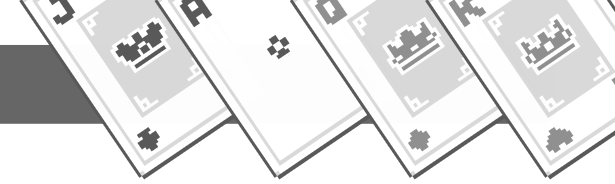
When in game, the player is able to see other players usernames and their chosen profile picture.

5. In-game UI
   a. Show the player the current community cards at the top of the screen.

   The current community cards are displayed directly on the top of the screen.

   b. Show the player the other bots/players partaking in the match.
      i. Each individual player's chips are drawn for the player.
      ii. Show the current status of a player.
      iii. If a player folds, text is displayed that they did so.
      iv. The last action is shown, for example, check, call £5, raise £7
      v. The current role of the player is displayed next to their displayed details with an icon. Showing whether they are a dealer, little blind, or big blind.

   The game displays the information about the player around the edge of the screen, including their name and image, the status of the player, their chips. The role is also shown next to their image.



c. Show the player the controls on the right side of the screen.
    i. Have text displayed alongside the button shown so that the user knows what the key controls what in game.
    ii. When the player increases or decreases their bet, it should be visible next to the betting controls.

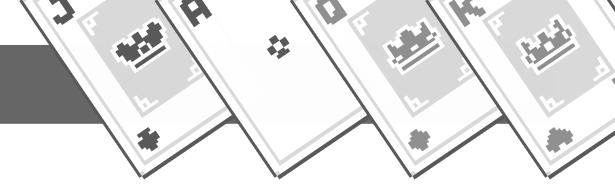
    The controls are in the bottom right corner of the screen and changes on the players actions.

d. Show the game information at the top of the screen.
    i. Showcase the time that game has been running for.
    ii. Show the game number and the turn number.
    iii. Display the name of the player who's go it is currently.

6. Game Logic
    a. During gameplay, be able to cycle between the main stages of the game: the pre-flop, the flop, the turn, the river, and the showdown.
        i. The 'round' is only progressed forward once everyone playing in that round has bet the same amount.
        ii. The pre-flop is the first round, no community cards are shown.
        iii. The flop is the second round, three community cards are shown.
        iv. The turn is the third round where fourth card is revealed.
        v. The river is the fourth round where the final community card is shown.
        vi. The showdown is the final round where players remain betting until they all bet the same.
            1. The player with the highest value hand wins the pot.
            2. Joint winners will have the pot split between winners.
        vii. Game is ended if all but one player folds.
        viii. Last remaining player wins the pot regardless.
        ix. Game will remain continuous unless the user decides to leave.

7. Bots
    a. Have a unique name and associated profile picture shown to other players.
    b. Able to bet and fold like other players.
        i. Decisions based on hand ranking, amount of money, money in the pot and an overall 'confidence level' assigned to each bot between 0-100 but capped at 10-90 so that the bot isn't overly aggressive or passive.
    c. Take a variable amount of time to make their decision so that it feels more natural to players instead of the choice being made instantly.

8. Poker Assistance
    a. Allow players to look at a table of hands during gameplay so that they can see what hands are possible and their ranking with percentage chance to obtain that hand.



b. A brief description on the rules and gameplay of Texas hold-em poker.
c. Table provided in game showing starting hand winning percentages to help newer players decide on how they would like to play with their hand.

9. Game Settings
    a. Allow the user to change a game setting which controls their username.
    b. Allow the user to change a game setting which controls which profile image is loaded when their character is loaded.
    c. Allow the user to enable and disable full screen setting.
    d. Allow the user to change the setting which controls the dimensions of the screen.

10. Art Style & Sound Design
    a. The game being called pixel poker means that the in-game assets and images should follow a pixel art style.
        i. As the user images can be chosen by individual players, these don't have to be in a pixel art style.

        The game assets apart from the profile pictures follow a pixel art style.

    b. Minimal sounds will be used in game.
        i. When pressing a button a sound should be played.
        ii. When increasing or decreasing your bet, a sound should be played.

        The game uses minimal sounds to communicate actions in game.

11. Communication
    a. Text chat in game to be able to communicate with other players.
        i. Ability to send text messages to other players within a multi-player game.
        ii. Should support Unicode characters.
        iii. Players are only able to type in the chat if they type in the message box.
        iv. Message is only sent when the enter key is pressed.
        v. Messages from other player should be shown in the chat.
        vi. Close or open chat.

        The text chat uses Unicode characters and can be closed or opened in game during a multiplayer game.

12. Save Data
    a. Save user settings under a text file.
        i. Data being username and profile picture plus settings.

b. When opening the game, the settings should be loaded.
    i. Using json.loads() to load the settings into the game.

    The game loads the players setting when loading, and saves any new changes such as name change.

Feedback

Client Feedback

Upon showing my clients my program and letting them use it here are what people enjoyed about the game.

- The simplicity of the menus made the game easy to use and navigate.
- The sound effects were minimal and give good feedback to the user about what is happening.
- The pixel art style is good and communicates what is happening in the game well.
- The settings were easy to change, and the user customisation is appreciated.
- Creating singleplayer games with bots is easy to do and quick.
- Creating multiplayer games and joining them with friends was easy.
- Seeing other players chips as well as yours and the money in the pot was a good touch.
- Actively needing to check my hand rather than have it constantly displayed made me feel more actively involved in the game.
- The poker assistance was useful in aiding my decisions whilst in game and made me feel more confident in game.

My thoughts and improvements

Obviously, due to this being the first version of my program, not everything in the program is perfect and could do with some improvements and changes. If I were to make any improvements, these are some I could make.

- The main menu could have been more visually interesting and take up more of the screen.
- On the text entry boxes allow the user to hold a key down to repeatedly enter it such as holding backspace rather than having to hold it down for every character.
- In the settings menu the changes that have been made could be more obvious, for example when writing in a text box in the settings menu, for the changes to save, the user must have to press enter for it to register.
- Remove or shorten the wait on the game start for single player games, reducing the amount of time a user would have to wait for a single player game.
- Make the join server menu easier to use, for example by allowing users to copy and paste the Ip address of the server in the server Ip box.
- During gameplay, prevent chips from either becoming very tall stacks and or prevent overlapping and clashing with other GUI elements such as the player chips and player name.

Improvements that could have been made if there was more time.

Naturally, if I had more time, I could have streamlined other part of the program making them work more effectively and or better. I could have also added more additional features etc. However, due to tight time constraints I had to focus on core gameplay, what I wrote in my objectives. In addition to tight time constraints, I needed to focus on the parts which would give me marks rather than stress over little details such as artwork tiny but feature which make the game feel more complete that would give me any extra marks.

- Adding a separate tutorial level would have been ideal as it would been able to explain to new poker players how to play the game in a safe and controlled environment where the player is able to be shown situations that they might experience in the real game at their own pace.

- Having globally save money and stats would have added a next level of depth, but due to the time constraints and the complexity of doing – needing to set up a server and then have the user communicate to the server to retrieve their player data. This would have allowed a competitive environment where players can compete to obtain the most money on a global leader board.
- Instead of the current system where the player joins a server by connecting to a multiplayer server by typing in the Ip of the server and port. It would have been better and easier to the user if they could see a list of open servers that are running and connect to the one, they like. However, doing this would require the need of a server to send to the user the Ip addresses of currently open servers. Otherwise, the user would have to ping every Ip address on the internet trying to find servers hosting 'Pixel Poker' servers.
- Allowing players who create servers to customise the settings on the server, such as starting number of credits for each player, wait time, ability to have bots on or off, kick other players from the server. Potentially, a lobby password so that the server host can control who is and isn't able to connect to the server.
- A in game VOIP (voice over ip) chat allowing people in the same server to communicate via voice, making the game feel more like at the table poker game rather than a game on the computer.

Appendix

Removed Data

Question 4 – Removed Responses

Losing lots of money. Getting kicked out for card counting
strip poker, real or virtual winnings
Cards
i would expect it to make me want to break my keyboard if i lose
make sure the colours are bright and the chips have detail
chips
Make sure you keep track of anything you need to
a good opponent so you don't always win too easily then maybe the camera on your computer looks at your face and if you smile the game picks it up and shows your in game character reacting well or negativly to your face reaction
fun
other people

Question 7 (Poker Players) – Removed Responses

No
Yes
not sure
i have played normal poker??
i didnt know there were different versions
black jack
[redacted] hold 'em (texas hold 'em but the gentlemens agreements are made up as we go along)
no and whats Texas Hold 'em? what are they holding?

Complete program code

main.py

```
import random, pygame, pygame.locals, os, traceback, time, math, compare,
client, server, json
from assetloader import *

try:
    with open('settings.txt', 'r') as f:
        settings = json.loads(f.read())
except:
    settings = {
        "Width": "1920",
        "Height": "1080",
        "Fullscreen": True,
        "Username": "Username",
        "Picture": "",
        "Server Port": "27015"
    }
    with open('settings.txt', 'w') as f:
        f.write(json.dumps(settings))


class Main():
    images = []

    def __init__(self):
        pygame.init()
        self.clock = pygame.time.Clock()
        self.info = pygame.display.Info()
        self.width, self.height = self.info.current_w, self.info.current_h
        self.size = [self.width, self.height]
        self.images = {}
        self.running = True
        self.sprites = pygame.sprite.Group()
        self.objects = []
        self.game = None

        self.Card = Card


        if settings["Fullscreen"]:
            self.window = pygame.display.set_mode(self.size, pygame.NOFRAME
| pygame.HWSURFACE | pygame.DOUBLEBUF)
        else:
            self.window = pygame.display.set_mode((int(settings['Width']),
int(settings['Height'])), pygame.RESIZABLE)
            self.size = [settings['Width'], settings['Height']]

        self.grey = (20, 20, 20)
        self.green = (15, 138, 19)

        self.load()
        self.client = client.Client(username=settings['Username'],
pfp=settings['Picture'], main=self)



        pygame.display.set_caption("Pixel Poker")
        pygame.display.set_icon(self.images['Poker icon'])
```

```
        self.mainMenu()

        self.loop()

    def updateSettings(self):
        with open('settings.txt', 'w') as f:
            f.write(json.dumps(settings))

    def writeSettings(self, param, data):
        settings[param] = data
        if param == 'Username':
            self.client.username = data
        elif param == 'Picture':
            self.client.pfp = data
        self.updateSettings()

    def updateFullscreen(self):
        if settings['Fullscreen']:
            self.writeSettings('Fullscreen', False)
        else:
            self.writeSettings('Fullscreen', True)

    def events(self):
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                self.running = False
            elif event.type == pygame.KEYDOWN:
                # if event.key == pygame.K_ESCAPE:
                #     self.quit()


                if self.game:
                    if self.client.turn:
                        if event.key == pygame.K_RETURN:
                            self.client.bet(self.client.betting)

            pos = pygame.mouse.get_pos()
            for Object in self.objects:
                if type(Object) == TextInput:
                    if event.type == pygame.MOUSEBUTTONDOWN:
                        if Object.rect.collidepoint(pos):
                            Object.active = True
                        else:
                            Object.active = False
                            Object.call_back()
                    elif event.type == pygame.KEYDOWN:
                        if event.key == pygame.K_BACKSPACE and
Object.active:

                            Object.textInput = Object.textInput[:-1]
                        elif event.key == pygame.K_RETURN and
Object.active:
                            Object.active = False
                            Object.call_back()
                        elif event.key == pygame.K_ESCAPE and
Object.active:
                            Object.active = False
                        elif Object.active and len(Object.textInput) <=
Object.charLimit:
                            Object.textInput += event.unicode
```

```
                elif type(Object) == Button:
                    if event.type == pygame.MOUSEBUTTONDOWN:
                        if event.button == 1:
                            buttonPressed = False
                            if Object.rect.collidepoint(pos) and not
buttonPressed:

pygame.mixer.Sound("Assets/Sound/Click.mp3").play()
                                Object.call_back()
                                buttonPressed = True

    def quit(self):
        self.running = False
        self.client.disconnect()

    def render(self):
        self.window.fill(self.background)
        self.sprites.draw(self.window)

    def removeAllSprites(self):
        self.sprites.empty()
        for object in self.objects:
            try:
                object.remove()
            except:
                object.remove2()
        self.objects = []

    def load(self):
        self.imagePaths = {}
        for (name, group) in imgGroups.items():
            for item in group:
                ext = ".png"
                splitted = item.split(".")
                if len(splitted) > 1:
                    item = splitted[0]
                    ext = "." + splitted[1]

                self.images[item] = pygame.image.load(os.path.join(name,
item + ext)).convert_alpha()
                self.imagePaths[item] = os.path.join(name, item + ext)


    def mainMenu(self):
        self.removeAllSprites()
        self.background = self.grey
        self.logo = Image(self, xy=[self.width/2 - 84, 30], dim=[169,95],
group=self.sprites, image=self.images['Logo'])
        self.b1 = Button(self, xy=[self.width/2, 150],
function=self.modeSelect, text='Play', dim=[200, 50], group=self.objects)
        self.b2 = Button(self, xy=[self.width/2, 200],
function=self.settingsMenu, text='Settings', dim=[200, 50],
group=self.objects)
        self.b3 = Button(self, xy=[self.width/2, 250], function=self.quit,
text='Quit', dim=[200, 50], group=self.objects)
        self.qr = Image(self, xy=[self.width/2, 350], dim=[58,58],
group=self.sprites, image=self.images['QR'])

    def settingsMenu(self):
        self.removeAllSprites()
        self.background = self.grey
```

```
        self.userText = TextLabel(self, text='Username:', xy=[self.width/2
- 180, 150], group=self.objects)
        self.pictureText = TextLabel(self, text='Picture:',
xy=[self.width/2 - 160, 186], group=self.objects)
        self.portText = TextLabel(self, text='Port:', xy=[self.width/2 -
140, 222], group=self.objects)
        self.widthText = TextLabel(self, text='Width:', xy=[self.width/2 -
150, 318], group=self.objects)
        self.heightText = TextLabel(self, text='Height:', xy=[self.width/2
- 150, 354], group=self.objects)
        self.t1 = TextInput(self, default=settings['Username'],
xy=[self.width/2 - 100, 150], function=self.writeSettings,
param='Username', group=self.objects)
        self.t2 = TextInput(self, default=settings['Picture'],
xy=[self.width/2 - 100, 186], function=self.writeSettings, param='Picture',
group=self.objects)
        self.t3 = TextInput(self, default=settings['Server Port'],
xy=[self.width/2 - 100, 222], function=self.writeSettings, param='Server
Port', group=self.objects)
        self.b1 = Button(self, xy=[self.width/2, 287],
function=self.updateFullscreen, text='Fullscreen Toggle' , dim=[250, 50],
group=self.objects)
        self.fScreentxt = TextLabel(self, text='(restart to take effect)',
xy=[self.width/2 + 290, 272], group=self.objects)
        self.t4 = TextInput(self, default=settings['Width'],
xy=[self.width/2 - 100, 318], function=self.writeSettings, param='Width',
group=self.objects)
        self.t5 = TextInput(self, default=settings['Height'],
xy=[self.width/2 - 100, 354], function=self.writeSettings, param='Height',
group=self.objects)
        self.b2 = Button(self, xy=[self.width/2, 419],
function=self.mainMenu, text='Back', dim=[200, 50], group=self.objects)

    def modeSelect(self):
        self.removeAllSprites()
        self.background = self.grey
        self.logo = Image(self, xy=[self.width/2 - 84, 30], dim=[169,95],
group=self.sprites, image=self.images['Logo'])
        self.b1 = Button(self, xy=[self.width/2, 150],
function=self.singleplayer, text='Singleplayer', dim=[200, 50],
group=self.objects)
        self.b2 = Button(self, xy=[self.width/2, 200],
function=self.joinOrCreate, text='Multiplayer', dim=[200, 50],
group=self.objects)
        self.b3 = Button(self, xy=[self.width/2, 250],
function=self.mainMenu, text='Back', dim=[200, 50], group=self.objects)
        self.qr = Image(self, xy=[self.width/2, 350], dim=[58,58],
group=self.sprites, image=self.images['QR'])


    def joinOrCreate(self):
        self.removeAllSprites()
        self.background = self.grey
        self.logo = Image(self, xy=[self.width/2 - 84, 30], dim=[169,95],
group=self.sprites, image=self.images['Logo'])
        self.b1 = Button(self, xy=[self.width/2, 150],
function=self.joinServerMenu, text='Join a lobby', dim=[200, 50],
group=self.objects)
        self.b2 = Button(self, xy=[self.width/2, 200],
function=self.createMultiplayerServer, text='Create a lobby', dim=[200,
50], group=self.objects)
```

```
        self.b3 = Button(self, xy=[self.width/2, 250],
function=self.modeSelect, text='Back', dim=[200, 50], group=self.objects)
        self.qr = Image(self, xy=[self.width/2, 350], dim=[58,58],
group=self.sprites, image=self.images['QR'])

    def joinServerMenu(self):
        self.removeAllSprites()
        self.background = self.grey
        self.userText = TextLabel(self, text='Server Ip:', xy=[self.width/2
- 185, 150], group=self.objects)
        self.portText = TextLabel(self, text='Port:', xy=[self.width/2 -
150, 186], group=self.objects)
        self.t1 = TextInput(self, default='', xy=[self.width/2 - 100, 150],
group=self.objects)
        self.t2 = TextInput(self, default='', xy=[self.width/2 - 100, 186],
group=self.objects)
        self.b1 = Button(self, xy=[self.width/2, 250],
function=self.connectToServer, text='Join', dim=[200, 50],
group=self.objects)
        self.b2 = Button(self, xy=[self.width/2, 300],
function=self.joinOrCreate, text='Back', dim=[200, 50], group=self.objects)


    def connectToServer(self):
        self.background = self.green
        self.connectionIp = self.t1.textInput
        self.connectionPort = self.t2.textInput
        self.removeAllSprites()
        self.game = Game(self, gametype="Multiplayer - J")

    def createMultiplayerServer(self):
        self.removeAllSprites()
        self.background = self.green
        self.game = Game(self, gametype="Multiplayer - C")

    def singleplayer(self):
        self.removeAllSprites()
        self.background = self.green
        self.game = Game(self, gametype="Singleplayer")


    def updateObjects(self):
        for Object in self.objects:
            Object.update()


    def loop(self):
        while self.running:
            self.dt = self.clock.tick(100) / 1000
            self.events()
            self.render()
            self.updateObjects()
            pygame.display.flip()
        pygame.quit()
        exit()

class Game():
    def __init__(self, parent, gametype=None, bots=None):
        self.parent = parent
        self.gametype = gametype
        self.players = []
```

```
        self.font = pygame.font.SysFont('Calibri',35)
        self.textColour = [0, 0, 0]
        self.gameStart = None

        self.pot = [0, 0, 0, 0, 0]
        self.communityCards = []
        self.lastbet = 0
        self.gameNumber = 0
        self.playerNumber = 0


        self.playerNumber = 0

        if self.gametype == "Singleplayer":
            self.server = server.Server(isInternal=True, game=self,
port=settings['Server Port'])
            connected = self.parent.client.connect('127.0.0.1',
settings['Server Port'])
        elif self.gametype == "Multiplayer - C":
            self.server = server.Server(game=self, port=settings['Server
Port'])
            connected = self.parent.client.connect('127.0.0.1',
settings['Server Port'])
        elif self.gametype == "Multiplayer - J":
            connected =
self.parent.client.connect(self.parent.connectionIp,
self.parent.connectionPort)
            self.server = False
        else:
            print('Attempted to create a gametype which isn\'t recognised')

        if not connected:
            print('Failed to connect.')
            return self.parent.mainMenu()


        self.mainPlayerHUD = MainPlayerUI(self, size=100,
picture=self.parent.client.pfp, username=self.parent.client.username)

    def setup(self):
        for player in self.players:
            player.setGame(self)

    def setLastBet(self, amount):
        self.lastbet = amount

    def getLastBet(self):
        return self.lastbet

    def getTurnNumber(self):
        return self.turnNumber

    def getGameNumber(self):
        return self.gameNumber

    def getPlayers(self):
        return self.players

    def getCommunityCards(self):
        return self.communityCards
```

```
class Card():
    def __init__(self, parent, suit, value, xy=[100, 100], size=[100,
100]):
        self.parent = parent
        self.suit = suit
        self.value = value
        self.xy = xy
        self.size = size
        self.values = {
            1 : "2",
            2 : "3",
            3 : "4",
            4 : "5",
            5 : "6",
            6 : "7",
            7 : "8",
            8 : "9",
            9 : "10",
            10 : "Jack",
            11 : "Queen",
            12 : "King",
            13 : "Ace"
        }
        if self.value:
            self.name = self.values[self.value] + " of " + self.suit
            self.card = self.parent.images[self.name]
        else:
            self.card = self.parent.images['card_empty']
            self.name = None
        self.back = self.parent.images['card_back']

        self.cardShown = False


    def switch(self):
        try:
            if self.value:
                if self.image.graphic == self.card:
                    self.image.update(self.back)
                else:
                    self.image.update(self.card)
            else:
                if self.image.graphic == self.card:
                    self.image.update(self.back)
                else:
                    self.image.update(self.parent.images['card_empty'])
        except:
            pass

    def getValue(self):
        return self.value

    def setValue(self, nValue):
        self.value = nValue

    def getName(self):
        return self.name
```

```
    def getValueName(self):
        return self.values[self.value]

    def setName(self):
        self.name = self.values[self.value] + " of " + self.suit

    def getSuit(self):
        return self.suit

    def setSuit(self, nSuit):
        self.suit = nSuit

    def getXY(self):
        return self.xy

    def setXY(self, nXY):
        self.xy = nXY

    def show(self):
        self.image = Image(self.parent, xy=self.xy, dim=self.size,
group=self.parent.sprites, image=self.card)
        self.cardShown = True

    def hide(self):
        self.image.kill()
        self.cardShown = False

    def remove(self):
        self.image.kill()
        del self


class Button():
    def __init__(self, parent, text='Button', font='Calibri', dim=[100,50],
xy=[0,0], textColour=[0, 0, 0], buttonColour=[122, 122, 122],
function=None, args=None, group=None):
        self.buttonText = text
        self.font = pygame.font.SysFont(font,35)
        self.buttonDim = dim
        self.xy = xy
        self.textColour = textColour
        self.buttonColour = buttonColour
        self.buttonActive = self.getButtonColours()
        self.function = function
        self.pressed = False
        self.parent = parent
        self.group = group
        self.args = args

        self.image = pygame.Surface((self.buttonDim[0], self.buttonDim[1]))
        self.rect = self.image.get_rect(center=(self.xy[0], self.xy[1]))

        self.text = self.font.render(self.buttonText, True,
self.textColour)

        self.group.append(self)

    def colourChange(self):
        self.buttonColour = self.buttonActive[0]
        pos = pygame.mouse.get_pos()
        if self.rect.collidepoint(pos):
```

```
            self.buttonColour = self.buttonActive[1]

    def update(self):
        self.colourChange()
        mouse = pygame.mouse.get_pos()
        self.image.fill((self.pressed and self.buttonActive[0]) or
self.buttonColour)
        self.image.blit(self.text, [self.rect.width/2 -
self.text.get_rect().width/2, self.rect.height/2 -
self.text.get_rect().height/2])
        self.parent.window.blit(self.image, self.rect)

    def getButtonColours(self):
        original = self.buttonColour
        highlight = []
        click = []
        for value in self.buttonColour:
            highlight.append(value + 50)
            click.append(value - 50)
        return [original, highlight, click]

    def call_back(self):
        self.buttonColour = self.buttonActive[2]
        if self.function:
            if self.args:
                self.function(self.args)
            else:
                self.function()

    def remove(self):
        global buttonList
        if self in self.group:
            self.group.remove(self)
        else:
            print('Im already gone')

class TextInput():
    def __init__(self, parent, font='Calibri', currentColour=(50, 50, 50),
selectColour=(100, 100, 100), default='', xy=[100, 100], size=[140, 35],
function=None, param=None, group=None):
        self.size = size
        self.xy = xy
        self.font = pygame.font.SysFont(font, self.size[1])
        self.textInput = str(default)
        self.rect = pygame.Rect(xy[0], xy[1], self.size[0], self.size[1])
        self.selectedColour = selectColour
        self.passiveColour = currentColour
        self.currentColour = self.passiveColour
        self.charLimit = 30
        self.active = False
        self.group = group
        self.parent = parent
        self.group.append(self)
        self.function = function
        self.param = param

    def update(self):
        if self.active:
            self.currentColour = self.selectedColour
        else:
            self.currentColour = self.passiveColour
```

```
        pygame.draw.rect(self.parent.window, self.currentColour, self.rect)
        inputtedText = self.font.render(self.textInput, True, (255, 255,
255))
        self.rect.w = max(100, inputtedText.get_width()+10)
        self.parent.window.blit(inputtedText, (self.rect.x + 5, self.rect.y
+ 5))

    def call_back(self):
        if self.function and self.param:
            self.function(self.param, self.textInput)
        elif self.function:
            self.function(self.textInput)

    def remove(self):
        if self in self.group:
            self.group.remove(self)
        else:
            print('Im already gone')


class Image(pygame.sprite.Sprite):
    def __init__(self, parent, image=None, xy=[0,0], dim=[100,100],
group=None):
        super().__init__()
        self.parent = parent
        self.graphic = image
        self.xy = xy
        if self.graphic != None:
            self.image = pygame.transform.scale(self.graphic, (dim[0],
dim[1]))
        else:
            self.graphic = self.parent.images['no image']
            self.image = pygame.transform.scale(self.graphic, (dim[0],
dim[1]))
        self.rect = self.image.get_rect(topleft=(self.xy[0], self.xy[1]))
        self.dim = dim
        self.group = parent.sprites


        self.group.add(self)

    def update(self, image):
        self.graphic = image
        if self.graphic != None:
            self.image = pygame.transform.scale(self.graphic, (self.dim[0],
self.dim[1]))
            self.rect = self.image.get_rect(topleft=(self.xy[0],
self.xy[1]))

    def remove2(self):
        if self in self.group:
            self.group.remove(self)
            self.kill()
        else:
            print('Im already gone')

class TextLabel():
    def __init__(self, parent, text='', font='Calibri', xy=[0,0],
textColour=(255, 255, 255), size=35, group=None):
        self.size = size
```

```
        self.font = pygame.font.SysFont(font, self.size)
        self.parent = parent
        self.text = text
        self.xy = xy
        self.group = group
        self.colour = textColour

        self.renderText = self.font.render(self.text, True, textColour)
        self.renderRect = self.renderText.get_rect()

        self.group.append(self)

    def update(self):
        self.parent.window.blit(self.renderText, (self.xy[0] -
self.renderRect.w/2, self.xy[1]), )

    def remove(self):
        if self in self.group:
            self.group.remove(self)
        else:
            print('Im already gone')


class PlayerInfo():
    def __init__(self, parent, player=None, xy=[100, 100], font='Calibri',
textColour=[0,0,0], size=100):
        self.parent = parent
        self.player = player
        self.xy = xy
        print(self.player)
        self.money = self.player['Money']
        self.username = self.player['Name']
        self.graphic = None
        self.group = self.parent.objects
        self.size = size
        self.chips = []
        self.roleIcon = None
        self.action = ''


        self.textColour = textColour
        self.font = pygame.font.SysFont(font, int(self.size * 0.35))
        self.text = self.font.render(self.username, True, self.textColour)

        self.profilePicture = Image(self.parent, image=self.graphic,
xy=self.xy, dim=[self.size, self.size])

        self.group.append(self)
        self.drawChips()
        self.drawRole()

    def drawChips(self):
        value = 1
        chipSpacingX = self.size / 4
        chipSpacingY = self.size / 50
        chipXY = [self.xy[0] + (self.size * 1.1),  self.xy[1] + (self.size
/ 2)]
        if self.chips != []:
            for chip in self.chips:
                chip.kill()
        self.chips = []
```

```
        for index in self.money:
            for x in range(index):
                self.chips.append(Image(self.parent,
image=self.parent.images[str(value)], xy=(chipXY[0], chipXY[1]),
dim=[self.size / 4, self.size / 4]))
                chipXY[1] -= chipSpacingY
            chipXY[0] += chipSpacingX
            chipXY[1] = self.xy[1] + (self.size / 2)
            value *= 2

    def drawRole(self):
        iconXY = [self.xy[0] + 100,  self.xy[1] + 100]
        if self.roleIcon != None:
            self.roleIcon.kill()
        playerRole = self.player['Role']
        if not playerRole == 'Player':
            if playerRole == 'Dealer':
                self.roleIcon = Image(self.parent,
image=self.parent.images['Dealer'], xy=(iconXY[0], iconXY[1]),
dim=[self.size / 4, self.size / 4])
            elif playerRole == 'Little Blind':
                self.roleIcon = Image(self.parent,
image=self.parent.images['Little blind'], xy=(iconXY[0], iconXY[1]),
dim=[self.size / 4, self.size / 4])
            elif playerRole == 'Big Blind':
                self.roleIcon = Image(self.parent,
image=self.parent.images['Big blind'], xy=(iconXY[0], iconXY[1]),
dim=[self.size / 4, self.size / 4])


    def update(self):
        self.actionText = self.font.render(self.action, True,
self.textColour)
        self.parent.window.blit(self.actionText, (self.xy[0] + (self.size *
1.1), self.xy[1] + 80), )
        if self.text:
            self.parent.window.blit(self.text, (self.xy[0] + (self.size *
1.1), self.xy[1]), )
        self.drawChips()
        self.drawRole()

    def remove2(self):
        if self in self.group:
            self.group.remove(self)
        else:
            print('Im already gone')



class MainPlayerUI():
    def __init__(self, parent, username='Username', picture=None,
hand=[None, None], font='Calibri', textColour=[0,0,0], size=100):
        self.parent = parent
        self.size = 100
        self.xy = [parent.parent.width/2 - self.size * 2.5, 3 *
parent.parent.height / 4]
        self.hand = []
        self.overallHand = []
        self.username = username
        try:
```

```
            self.graphic = pygame.image.load('./User Images/' +
picture).convert_alpha()
        except:
            self.graphic = self.parent.parent.images['no image']
        self.group = self.parent.parent.objects
        self.group.append(self)
        self.handShown = 0
        self.chips = []
        self.potChips = []
        self.emptyCards = []
        self.roleIcon = None
        self.role = 'Player'
        self.totalMoney = 0
        self.playerDisplays = {}

        self.textColour = textColour
        self.font = pygame.font.SysFont(font, int(self.size * 0.35))

        self.bettingAmount = 0
        self.action = "Check"
        self.foldAction = "Fold"
        self.handAction = "Check Hand"
        self.chatText = 'Toggle Chat'
        self.handName = ''
        self.timeElapsed = '0'
        self.foldActionText = self.font.render(self.foldAction, True,
self.textColour)
        self.handActionText = self.font.render(self.handAction, True,
self.textColour)
        self.pokerHelp = False
        self.pokermenu = None
        self.paused = False
        self.lastTime = 0
        self.lastTime2 = 0
        self.b1 = None
        self.textInput = None
        self.chatMessages = ['mate', 'john', 'egg']
        self.chatting = False
        self.messages = []
        self.textObjects = []

        self.drawSelfPlayerInfo()
        self.drawControls()
        self.drawHand()
        self.drawPlayers()
        self.drawCommunityCards()
        self.drawRole()

    def drawPokerHelpMenu(self):
        if self.pokerHelp:
            if not self.pokermenu:
                self.pokermenu = Image(self.parent.parent,
image=self.parent.parent.images['Poker Assistance'],
dim=[self.parent.parent.width/2, self.parent.parent.height],
xy=[self.parent.parent.width/2 - self.parent.parent.width/4, 0])
        elif self.pokermenu:
            self.pokermenu.remove2()
            self.pokermenu = None

    def drawSelfPlayerInfo(self):
```

```
        self.profilePicture = Image(self.parent.parent, image=self.graphic,
xy=self.xy, dim=[self.size, self.size])
        self.text = self.font.render(self.username, True, self.textColour)


    def drawChips(self):
        value = 1
        chipSpacingX = self.size / 1.5
        chipSpacingY = 4 * self.size / 80
        chipXY = [self.xy[0] + (self.size * 1.1),  self.xy[1] + (self.size
/ 1.5)]
        if self.chips != []:
            for chip in self.chips:
                chip.kill()
        self.chips = []
        for index in self.parent.parent.client.money:
            for x in range(index):
                self.chips.append(Image(self.parent.parent,
image=self.parent.parent.images[str(value)], xy=(chipXY[0], chipXY[1]),
dim=[self.size / 1.5, self.size / 1.5]))
                chipXY[1] -= chipSpacingY
            chipXY[0] += chipSpacingX
            chipXY[1] = self.xy[1] + (self.size / 1.5)
            value *= 2

    def calculatePot(self):
        value = 1
        totalMoney = 0
        for i in self.parent.pot:
            amount = value * i
            totalMoney += amount
            value *= 2
        return totalMoney

    def drawPot(self):
        value = 1
        chipSpacingX = self.size / 1.5
        chipSpacingY = 4 * self.size / 80
        chipY = self.parent.parent.height / 2
        chipXY = [self.parent.parent.width/2 - self.size * 2,  chipY]
        if self.potChips != []:
            for chip in self.potChips:
                chip.kill()
        self.potChips = []
        for index in self.parent.pot:
            for x in range(index):
                self.potChips.append(Image(self.parent.parent,
image=self.parent.parent.images[str(value)], xy=(chipXY[0], chipXY[1]),
dim=[self.size / 1.5, self.size / 1.5]))
                chipXY[1] -= chipSpacingY
            chipXY[0] += chipSpacingX
            chipXY[1] = chipY
            value *= 2

        self.potValue = self.font.render('Pot: £' +
str(self.calculatePot()), True, self.textColour)
        self.potVRect = self.potValue.get_rect()

    def drawRole(self):
        iconXY = [self.xy[0] + 100,  self.xy[1] + 100]
        if self.roleIcon != None:
```

```
            self.roleIcon.kill()
        playerRole = self.role
        if not playerRole == 'Player':
            if playerRole == 'Dealer':
                self.roleIcon = Image(self.parent.parent,
image=self.parent.parent.images['Dealer'], xy=(iconXY[0], iconXY[1]),
dim=[self.size / 4, self.size / 4])
            elif playerRole == 'Little Blind':
                self.roleIcon = Image(self.parent.parent,
image=self.parent.parent.images['Little blind'], xy=(iconXY[0], iconXY[1]),
dim=[self.size / 4, self.size / 4])
            elif playerRole == 'Big Blind':
                self.roleIcon = Image(self.parent.parent,
image=self.parent.parent.images['Big blind'], xy=(iconXY[0], iconXY[1]),
dim=[self.size / 4, self.size / 4])

    def drawControls(self):
        self.xpos = self.parent.parent.width
        self.ypos = self.parent.parent.height - 160
        self.upArrow = Image(self.parent.parent,
image=self.parent.parent.images['Up'],xy=[self.xpos - 40, self.ypos] ,
dim=[40, 40])
        self.downArrow = Image(self.parent.parent,
image=self.parent.parent.images['Down'], xy=[self.xpos - 80, self.ypos],
dim=[40, 40])
        self.enter = Image(self.parent.parent,
image=self.parent.parent.images['Enter'], xy=[self.xpos - 40, self.ypos +
40], dim=[40, 40])
        self.fKey = Image(self.parent.parent,
image=self.parent.parent.images['F'], xy=[self.xpos - 40, self.ypos + 80],
dim=[40, 40])
        self.spacebar = Image(self.parent.parent,
image=self.parent.parent.images['Space'], xy=[self.xpos - 40, self.ypos +
120], dim=[40, 40])
        self.tabkey = Image(self.parent.parent,
image=self.parent.parent.images['Tab'], xy=[self.xpos - 40, self.ypos -
40], dim=[40, 40])
        if self.parent.server:
            if not self.parent.server.isInternal:
                self.altkey = Image(self.parent.parent,
image=self.parent.parent.images['Alt'], xy=[self.xpos - 40, self.ypos -
80], dim=[40, 40])
        else:
            self.altkey = Image(self.parent.parent,
image=self.parent.parent.images['Alt'], xy=[self.xpos - 40, self.ypos -
80], dim=[40, 40])

    def drawHand(self):
        if len(self.hand) >= 2:
            card1 = self.hand[0]
            card2 = self.hand[1]
            card1.setXY([self.parent.parent.width/2 - 100,
self.parent.parent.height - 100])
            card2.setXY([self.parent.parent.width/2 - 30,
self.parent.parent.height - 100])
            card1.show()
            card2.show()
            card1.switch()
            card2.switch()

    def drawPlayers(self):
```

```
        players = self.parent.players
        playerIndex = 0
        for idx, val in enumerate(players):
            if val['Id'] == self.parent.parent.client.id:
                playerIndex = idx
                break
        visPlayers = players[playerIndex:] + players[:playerIndex]
        playerPositions = [
            [self.parent.parent.width/20, self.parent.parent.height/2],
            [self.parent.parent.width/20, self.parent.parent.height/12],
            [self.parent.parent.width - self.parent.parent.width/5,
self.parent.parent.height/12],
            [self.parent.parent.width - self.parent.parent.width/5,
self.parent.parent.height/2],
        ]

        self.playerDisplays = {}

        for playerNumber in range(len(visPlayers)):
            if visPlayers[playerNumber]['Id'] !=
self.parent.parent.client.id:
                playerObject = PlayerInfo(self.parent.parent,
player=visPlayers[playerNumber], xy=playerPositions[playerNumber - 1])
                self.playerDisplays[visPlayers[playerNumber]['Id']] =
playerObject


    def flipHand(self):
        self.cardXY = [self.parent.parent.width/2 - 70,
self.parent.parent.height - 100]
        if self.handShown == 0:
            self.handShown = 1
        else:
            self.handShown = 0
        for card in self.hand:
            card.switch()

    def drawCommunityCards(self):
        self.cardsxy = [self.parent.parent.width /2 - 190, 70]
        for card in self.parent.communityCards:
            card.setXY(self.cardsxy)
            card.show()
            self.cardsxy[0] += 70
        for card in range(5 - len(self.parent.communityCards)):
            blankCard = Card(self.parent.parent, None, None)
            blankCard.setXY(self.cardsxy)
            blankCard.show()
            blankCard.switch()
            self.emptyCards.append(blankCard)
            self.cardsxy[0] += 70

    def removeCommunityCards(self):
        for blank in self.emptyCards:
            blank.remove()
        self.emptyCards = []
        for card in self.parent.communityCards:
            if card.cardShown:
                card.hide()

    def gameInfo(self):
        if self.parent.parent.client.currentPlayer != None:
```

```
            self.currentPlayerText =
self.font.render(self.parent.parent.client.currentPlayer + "'s Turn", True,
self.textColour)
        else:
            self.currentPlayerText = self.font.render("Game is starting",
True, self.textColour)
        self.parent.parent.window.blit(self.currentPlayerText,
(self.parent.parent.width/2 - self.currentPlayerText.get_rect().width/2,
40))

        self.timeText = self.font.render(self.getTime() + ' Game 1, Turn
1', True, self.textColour)
        self.timeRect = self.timeText.get_rect()
        self.parent.parent.window.blit(self.timeText,
(self.parent.parent.width/2 - self.timeRect.width/2, 0))

    def getTime(self):
        if self.parent.gameStart:
            self.timeInS = math.floor(time.time() - self.parent.gameStart)
            self.minutesElapsed = math.floor(self.timeInS/60)
            self.hoursElapsed = math.floor(self.minutesElapsed/60)

            self.timer = f"{self.hoursElapsed:02}:{self.minutesElapsed -
(self.hoursElapsed * 60):02}:{self.timeInS - (self.minutesElapsed *
60):02}"
            return self.timer
        else:
            return 'Waiting For Start,'

    def togglePause(self):
        if not self.paused and time.time() - self.lastTime >= 0.1:
            self.lastTime = time.time()
            self.paused = True
        elif time.time() - self.lastTime >= 2:
            self.lastTime = time.time()
            self.b1.remove()
            self.b1 = None
            self.b2.remove()
            self.b2 = None
            self.paused = False

    def toggleChat(self):
        if not self.chatting and time.time() - self.lastTime2 >= 0.1:
            self.lastTime2 = time.time()
            self.chatting = True
        elif time.time() - self.lastTime2 >= 2:
            self.lastTime2 = time.time()
            self.textInput.remove()
            for i in self.textObjects:
                i.remove()
            self.textObjects = []
            self.textInput = None
            self.chatting = False

    def leave(self):
        self.parent.parent.client.disconnect()
        self.parent.parent.game = None
        self.parent.parent.mainMenu()

    def sendMessage(self, text):
        print('Yes you called')
```

```
        if self.textInput and text != '':
            self.parent.parent.client.networkQueue.put(b'7['+ bytes(text,
'utf-8') + b']\x1e')
            self.textInput.textInput = ''

    def chatMenu(self):
        positions = [
            [150, self.parent.parent.height - 135],
            [150, self.parent.parent.height - 170],
            [150, self.parent.parent.height - 205],
            [150, self.parent.parent.height - 240],
            [150, self.parent.parent.height - 275]
        ]
        pygame.draw.rect(self.parent.parent.window, (20, 20, 20),
pygame.Rect(50, self.parent.parent.height - 300, 270, 200))
        if not self.textInput:
            self.textInput = TextInput(self.parent.parent,
function=self.sendMessage, param='', xy=[50, self.parent.parent.height -
100], group=self.parent.parent.objects)
        for i in self.textObjects:
            i.remove()
        self.textObjects = []
        count = 0
        for message in self.chatMessages:
            self.textObjects.append(TextLabel(self.parent.parent,
text=message, xy=positions[count], group=self.parent.parent.objects))
            count += 1


    def pauseMenu(self):
        pygame.draw.rect(self.parent.parent.window, (20, 20, 20),
pygame.Rect(0, 0, self.parent.parent.width, self.parent.parent.height))
        if not self.b1 or not self.b2:
            self.b1 = Button(self.parent.parent, text='Quit',
function=self.parent.parent.quit,  xy=[self.parent.parent.width/2,
self.parent.parent.height/2],group=self.parent.parent.objects)
            self.b2 = Button(self.parent.parent, text='Main Menu',
function=self.leave, dim=[250, 50], xy=[self.parent.parent.width/2,
self.parent.parent.height/2 - 50],group=self.parent.parent.objects)


    def update(self):
        self.overallHand = self.hand + self.parent.communityCards

        self.drawPokerHelpMenu()
        self.drawPot()
        self.drawChips()
        self.gameInfo()
        self.removeCommunityCards()
        self.drawCommunityCards()
        self.drawRole()

        self.betText = self.font.render(("£" +
str(self.parent.parent.client.betting)), True, self.textColour)
        self.pokerAssistance = self.font.render("Poker Assistance", True,
self.textColour)
        self.textChat = self.font.render(self.chatText, True,
self.textColour)
        if self.parent.lastbet < self.parent.parent.client.betting:
            self.betPromptText = self.font.render("Raise", True,
self.textColour)
```

```
        elif self.parent.parent.client.betting == 0:
            self.betPromptText = self.font.render("Check", True,
self.textColour)
        elif self.parent.lastbet == self.parent.parent.client.betting:
            self.betPromptText = self.font.render("Call", True,
self.textColour)
        elif self.parent.parent.client.betting ==
self.parent.parent.client.betting:
            self.betPromptText = self.font.render("All in", True,
self.textColour)
        self.betTextRect = self.betText.get_rect()
        self.betPromptRect = self.betPromptText.get_rect()
        self.parent.parent.window.blit(self.text, (self.xy[0] + (self.size
* 1.1), self.xy[1]))
        self.parent.parent.window.blit(self.betText, (self.xpos -
self.betTextRect.width - 85, self.ypos))
        self.parent.parent.window.blit(self.pokerAssistance, (self.xpos -
280, self.ypos - 40))
        self.parent.parent.window.blit(self.betPromptText, (self.xpos -
self.betPromptRect.width - 45, self.ypos + 40))
        self.parent.parent.window.blit(self.foldActionText, (self.xpos -
105, self.ypos + 80))
        self.parent.parent.window.blit(self.handActionText, (self.xpos -
210, self.ypos + 120))
        self.parent.parent.window.blit(self.potValue,
(self.parent.parent.width - self.potVRect.width, 0))
        if self.parent.server:
            if not self.parent.server.isInternal:
                self.parent.parent.window.blit(self.textChat, (self.xpos -
280, self.ypos - 80))
        else:
            self.parent.parent.window.blit(self.textChat, (self.xpos - 280,
self.ypos - 80))
        if self.handShown == 1:
            self.handValue = compare.getValueOfHand(self.overallHand)
            self.handName = compare.valueToName(self.handValue)
            self.handNameText = self.font.render(self.handName, True,
self.textColour)
            self.handNameTextRect = self.handNameText.get_rect()
            self.parent.parent.window.blit(self.handNameText,
(self.parent.parent.width/2 - (self.handNameTextRect.width /2),
(self.parent.parent.height * 9/10) - self.handNameTextRect.height))


        fpsText =
self.font.render(str(int(self.parent.parent.clock.get_fps())), True,
self.textColour)
        if self.parent.server:
            if self.parent.server.isInternal:
                serverInfoText = self.font.render("Singleplayer", True,
self.textColour)
            else:
                serverInfoText = self.font.render("Server Ip: " +
self.parent.server.globalIp + ":" + str(self.parent.parent.client.port),
True, self.textColour)
        else:
            serverInfoText = self.font.render("Server Ip: " +
self.parent.parent.client.serverIp + ":" +
str(self.parent.parent.client.port), True, self.textColour)
        serverInfoTextRect = serverInfoText.get_rect()
        self.parent.parent.window.blit(fpsText, (0, 0))
```

```
        self.parent.parent.window.blit(serverInfoText, (0,
self.parent.parent.height - serverInfoTextRect.height))

        if self.handShown == 1:
            self.flipHand()
        keys = pygame.key.get_pressed()
        if keys[pygame.K_SPACE]: self.flipHand()
        if keys[pygame.K_ESCAPE]: self.togglePause()
        if keys[pygame.K_LALT]: self.toggleChat()
        if keys[pygame.K_p]:
self.parent.parent.client.networkQueue.put(b'7["Mr Margon"]\x1e')
        if keys[pygame.K_TAB]:
            self.pokerHelp = True
        else:
            self.pokerHelp = False
        if self.parent.parent.client.turn:
            if keys[pygame.K_UP]:
                self.parent.parent.client.increaseBet()
            elif keys[pygame.K_DOWN]:
                self.parent.parent.client.decreaseBet()
            elif keys[pygame.K_f]:
                self.parent.parent.client.betting = 0
                self.parent.parent.client.fold()

        if self.paused:
            self.pauseMenu()
        if self.chatting:
            self.chatMenu()

    def remove(self):
        for i in self.potChips:
            i.remove()
        for i in self.chips:
            i.remove()
        self.removeCommunityCards()



if __name__ == '__main__':
    m = Main()

pygame.quit()
```

compare.py

```
from functools import lru_cache


def sortRule(x):
    return x.value

def concatenateInts(a, b):
    return a*100 + b


def getValueOfHand(hand):
    if len(hand) < 2:
        return 0

    # hand value = 01-10 + 01-13 e.g. 0510 = jack high straight

    hand.sort(key=sortRule)
    concurrent = 0
    straight = 0
    suitCount = [0, 0, 0, 0]
    pairCount = [0, 0, 0] # pair, three, four
    suits = ["Clubs", "Diamonds", "Hearts", "Spades"]
    prevCard = None
    hands = []
    hasStraight = False
    hasFlush = False
    highestStraightCard = None
    highestPairCard = None
    for i in range(len(hand)):
        if not i == 0:

            # check for pair, two pair, three of a kind, and four of a kind
            if prevCard.value == hand[i].getValue():
                concurrent += 1

            if concurrent >= 1 and not prevCard.getValue() ==
hand[i].getValue():
                pairCount[concurrent - 1] += 1
                concurrent = 0
                highestPairCard = prevCard
            elif not prevCard.getValue() == hand[i].getValue():
                concurrent = 0

            # check for straight - 05
            if prevCard.getValue() + 1 == hand[i].getValue():
                straight += 1

            if straight >= 4 and not prevCard.getValue() + 1 ==
hand[i].getValue():
                highestStraightCard = prevCard
                hasStraight = False
                straight = 0
            elif not prevCard.getValue() + 1 == hand[i].getValue():
                straight = 0

        prevCard = hand[i]

        #check for flush
        suitCount[suits.index(hand[i].getSuit())] += 1
```

```
    # get high card - 01
    highCard = hand[len(hand) - 1]
    value = concatenateInts(1, highCard.getValue())
    hands.append(value)
#     print(highCard.getValueName(), 'high')

    # print pairs - 02, 03, 04, 08
    for i in range(len(pairCount)):
        if i == 0:
            if pairCount[i] == 1:
                value = concatenateInts(2, highestPairCard.getValue())
                hands.append(value)
#                 print("Pair of " + highestPairCard.getValueName() + "'s")
            elif pairCount[i] == 2:
                value = concatenateInts(3, highestPairCard.getValue())
                hands.append(value)
#                 print("Two Pair " + highestPairCard.getValueName() +
"'s")
        elif i == 1:
            if pairCount[i] == 1:
                value = concatenateInts(4, highestPairCard.getValue())
                hands.append(value)
#                 print("Three of a kind " + highestPairCard.getValueName()
+ "'s")
        elif i == 2:
            if pairCount[i] == 1:
                value = concatenateInts(8, highestPairCard.getValue())
                hands.append(value)
#                 print("Four of a kind " + highestPairCard.getValueName()
+ "'s")

    # check for full house - 07
    if pairCount[0] >= 1 and pairCount[1] >= 1:
        value = concatenateInts(7, highestPairCard.getValue())
        hands.append(value)
#         print("Full house " + highestPairCard.getValueName() + " high")

    # print flush - 06
    for i in range(len(suitCount)):
        if suitCount[i] >= 5:
            for card in hand:
                if card.getSuit() == suits[i]:
                    bestCard = card
            value = concatenateInts(6, bestCard.getValue())
            hands.append(value)
            hasFlush = True
#             print(suit[i] ,'Flush')


    if hasStraight:
        if hasFlush:
            value = concatenateInts(9, highestStraightCard.getValue())
            hands.append(value)
#             print(highestStraightCard.getValueName(), 'Flush High
Straight')
        else:
            value = concatenateInts(5, highestStraightCard.getValue())
            hands.append(value)
#             print(highestStraightCard.getValueName(), 'High Straight')

    return max(hands)
```

```
def valueToName(value):
    hands = {
        1 : "? High",
        2 : "Pair of ?'s",
        3 : "Two Pair ?'s",
        4 : "Three of a Kind",
        5 : "? High Straight",
        6 : "? High Flush",
        7 : "Full House",
        8 : "Four of a Kind",
        9 : "? High Straight Flush",
    }

    cardNames = {
            1 : "2",
            2 : "3",
            3 : "4",
            4 : "5",
            5 : "6",
            6 : "7",
            7 : "8",
            8 : "9",
            9 : "10",
            10 : "Jack",
            11 : "Queen",
            12 : "King",
            13 : "Ace"
        }

    if value == 0:
        return 'No Hand'

    handNumber, cardName = int(str(value)[:1]),
cardNames[int(str(value)[1:])]

    if value != 913:
        handName = hands[handNumber].replace("?", cardName)
    else:
        handName = "Royal Flush"

    return handName
```

assetloader.py

```
import os

imgGroups = {
    "Assets/Cards/": [
            '2 of Hearts',
            '3 of Hearts',
            '4 of Hearts',
            '5 of Hearts',
            '6 of Hearts',
            '7 of Hearts',
            '8 of Hearts',
            '9 of Hearts',
            '10 of Hearts',
            'Jack of Hearts',
            'Queen of Hearts',
            'King of Hearts',
            'Ace of Hearts',
            '2 of Diamonds',
            '3 of Diamonds',
            '4 of Diamonds',
            '5 of Diamonds',
            '6 of Diamonds',
            '7 of Diamonds',
            '8 of Diamonds',
            '9 of Diamonds',
            '10 of Diamonds',
            'Jack of Diamonds',
            'Queen of Diamonds',
            'King of Diamonds',
            'Ace of Diamonds',
            '2 of Clubs',
            '3 of Clubs',
            '4 of Clubs',
            '5 of Clubs',
            '6 of Clubs',
            '7 of Clubs',
            '8 of Clubs',
            '9 of Clubs',
            '10 of Clubs',
            'Jack of Clubs',
            'Queen of Clubs',
            'King of Clubs',
            'Ace of Clubs',
            '2 of Spades',
            '3 of Spades',
            '4 of Spades',
            '5 of Spades',
            '6 of Spades',
            '7 of Spades',
            '8 of Spades',
            '9 of Spades',
            '10 of Spades',
            'Jack of Spades',
            'Queen of Spades',
            'King of Spades',
            'Ace of Spades',
            'card_back',
            'card_empty',
        ],
    "Assets/Default Pfp/": [
```

```
            "gameoil",
            "teams",
            "no image",
            "Graham",
            "Lukas",
            "Bahn",
            "Jim",
            "Jorge",
            "Kallum",
            "Sachet",
            "Red Chief",
            "John",
            "Greg",
            "Joe",
            "Margaret",
            "Barbara",
            'Agatha',
            'Helena',
            'Christine',
            'Jenny'
        ],
    "Assets/Chips/": [
            '1',
            '2',
            '4',
            '8',
            '16',
            'Big blind',
            'Little blind',
            'Dealer',
            'Poker icon',
        ],
    "Assets/Keys/": [
            'Alt',
            'Down',
            'Enter',
            'F',
            'Space',
            'Up',
            'Tab',
        ],
    "Assets/Misc/": [
            'Poker Assistance',
        ],
    "Assets/": ["QR", "Logo"]
    }

imagePaths = {}

def load():
    for (name, group) in imgGroups.items():
        for item in group:
            ext = ".png"
            splitted = item.split(".")
            if len(splitted) > 1:
                item = splitted[0]
                ext = "." + splitted[1]

            imagePaths[item] = os.path.join(name, item + ext)
load()
```

client.py

```
import socket, os, json, threading, pygame, base64, assetloader, time,
math, queue

packetID = {
    '0' : 'Connect',
    '1' : 'Disconnect',
    '2' : 'Player Data',
    '3' : 'New Game',
    '4' : 'Picture',
    '5' : 'Id',
    '6' : 'Loaded',
    '7' : 'Message',
    '8' : 'Start Game',
    '9' : 'Turn',
    '10' : 'Bet',
    '11' : 'Fold',
    '1000' : 'Ping',
}


#change port back to 6778

class Client():
    def __init__(self, username, pfp='', main=None):
        self.username = username
        self.pfp = pfp
        self.main = main

        self.client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.client.settimeout(1)
        self.connection = None

        self.turn = False
        self.betting = 0
        self.money = [0, 0, 0, 0, 0]
        self.values = [1, 2, 4, 8, 16]
        self.totalMoney = 0
        self.currentPlayer = None
        self.id = None
        self.networkQueue = queue.Queue()
        self.folded = False

        self.stopping = threading.Event()
        self.timeDebounce = 0


    def serialiseClient(self):
        data = {
            'Name' : self.username,
        }

        return json.dumps(data)

    def sendingImages(self, picture):
        try:
            with open('./User Images/' + self.pfp, 'rb') as f:
                convertedImage = base64.b64encode(f.read())
                self.client.sendall((convertedImage + b'\x1e'))
        except:
            with open(assetloader.imagePaths['no image'], 'rb') as f:
```

```
                convertedImage = base64.b64encode(f.read())
                self.client.sendall((convertedImage + b'\x1e'))

    def connect(self, serverIp, port):
        self.serverIp = serverIp
        self.port = int(port != ''  and port or 27015)
        try:
            self.client.connect((self.serverIp, self.port))
            threading.Thread(target=self.clientLoop, args=(),
daemon=True).start()
            return True
        except Exception as e:
            print("Connection error:", e)
            return False

    def send(self, data):
        try:
            self.client.sendall((data + '\x1e').encode())
        except Exception as e:
            print('Sending error:', e)

    def recieveData(self):
        data = b''
        while not self.stopping.is_set():
            try:
                recv = self.client.recv(4096)
                data += recv
                if not recv:
                    break
                elif recv[-1:] == b'\x1e':
                    data = data[:-1]
                    break
            except Exception as e:
                break
            except:
                break
        packetId = ''
        while len(data) > 0 and chr(data[0]).isnumeric():
            packetId += chr(data[0])
            data = data[1:]
        if packetId == '':
            return '1000', None
        return packetId, data

    def clientLoop(self):
        self.send(self.serialiseClient())
        self.sendingImages(self.pfp)
        while not self.stopping.is_set():
            packetId, data = self.recieveData()
            match packetID[packetId]:
                case "Connect":
                    pass
                case "Disconnect":
                    pass
                case "Id":
                    data = json.loads(data.decode())
                    self.id = data[0]
                case "Player Data":
                    print(data)
                    data = json.loads(data.decode())
                    self.main.game.players = data['Players']
```

```
                    self.turn = data['Current Player'] and (data['Current
Player']['Id'] == self.main.client.id)

                    if
len(self.main.game.mainPlayerHUD.playerDisplays.keys()) > 0:
                        for idx, val in enumerate(data['Players']):
                            if (val['Id'] != self.main.client.id):

self.main.game.mainPlayerHUD.playerDisplays[val['Id']].money = val['Money']

self.main.game.mainPlayerHUD.playerDisplays[val['Id']].action = val['Last
Action']

self.main.game.mainPlayerHUD.playerDisplays[val['Id']].username =
val['Name']

                    self.main.game.mainPlayerHUD.hand = [
                        self.main.Card(self.main, data['Hand'][0]['suit'],
data['Hand'][0]['value']),
                        self.main.Card(self.main, data['Hand'][1]['suit'],
data['Hand'][1]['value'])
                    ]
                    self.main.game.communityCards = []
                    for card in data['Community Cards']:

self.main.game.communityCards.append(self.main.Card(self.main,
card['suit'], card['value']))
                    self.main.game.mainPlayerHUD.drawHand()

                    if data['Current Player']:
                        self.currentPlayer = data['Current Player']['Name']

                    print(data)
                    self.main.client.money = data['Money']
                    self.main.game.mainPlayerHUD.role = data['Role']
                    self.main.game.pot = data['Pot']
                    self.main.game.lastbet = data['Last Bet']
                case "New Game":
                    for display in
self.main.game.mainPlayerHUD.playerDisplays.values():
                        display.remove2()
                    self.main.game.mainPlayerHUD.playerDisplays = {}
                    self.main.game.mainPlayerHUD.drawPlayers()
                    for clientId, playerInfo in
self.main.game.mainPlayerHUD.playerDisplays.items():
                        if os.path.exists('./pfp/' +  clientId + '.png'):

playerInfo.profilePicture.update(pygame.image.load('./pfp/' + clientId +
'.png'))
                    self.networkQueue.put(b'6[]\x1e')
                case "Picture":
                    data = data[1:]
                    clientId = ''
                    while len(data) > 0 and chr(data[0]).isnumeric():
                        clientId += chr(data[0])
                        data = data[1:]
                    data = data[1:]
                    picture = base64.b64decode(data + b'==')
                    with open('./pfp/' + clientId + '.png', 'wb') as f:
                        f.write(picture)
```

```
                    for cid, playerInfo in
self.main.game.mainPlayerHUD.playerDisplays.items():
                        if os.path.exists('./pfp/' +  clientId + '.png')
and cid == clientId:

playerInfo.profilePicture.update(pygame.image.load('./pfp/' + clientId +
'.png'))
                case "Message":
                    data = str(data.decode('utf-8'))
                    print(data)
                    if len(self.main.game.mainPlayerHUD.chatMessages) <= 4:

self.main.game.mainPlayerHUD.chatMessages.append(data[0])
                    else:
                        self.main.game.mainPlayerHUD.chatMessages.pop(0)

self.main.game.mainPlayerHUD.chatMessages.append(data[0])
                case "Turn":
                    self.main.client.turn = True
                    pygame.mixer.Sound("Assets/Sound/Ping.mp3").play()
                    self.totalMoney = self.calculateMoney()
                    self.betting = self.main.game.lastbet
                case "Start Game":
                    data = json.loads(data)
                    self.main.game.gameStart = data['Start Time']
                case _:
                    pass
            if not self.stopping.is_set():
                if not self.networkQueue.empty():
                    self.client.sendall(self.networkQueue.get())
                else:
                    self.client.sendall(b'1000[]\x1e')

    def minAmount(self):
        for idx, val in enumerate(self.money):
            if val >= 1:
                return self.values[idx]


    def increaseBet(self):
        minAmount = self.minAmount()
        if time.time() - self.timeDebounce >= 0.1:
            self.timeDebounce = time.time()
            if not self.betting + 1 > self.totalMoney:
                self.betting += minAmount

    def decreaseBet(self):
        minAmount = self.minAmount()
        if time.time() - self.timeDebounce >= 0.1:
            self.timeDebounce = time.time()
            if not self.betting - 1 < 0 and not self.betting - 1 <
self.main.game.getLastBet():
                self.betting -= minAmount

    def fold(self):
        self.folded = True
        self.betting = 0
        self.endGo()

    def endGo(self):
        if self.folded:
```

```
            self.networkQueue.put(b'11[]\x1e')
        else:
            self.networkQueue.put(bytes(f'10[{self.betting}]\x1e', 'utf-
8'))
        self.turn = False
        self.betting = 0

    def calculateMoney(self):
        value = 1
        totalMoney = 0
        for i in self.money:
            amount = value * i
            totalMoney += amount
            value *= 2
        self.totalMoney = totalMoney
        return self.totalMoney

    def bet(self, amount):
        starting = amount
        chipsAdded = [0, 0, 0, 0, 0]
        print(amount)
        if not amount == 0:
            for i in range(4, -1, -1):
                print(amount, self.values[i])
                if amount >= self.values[i] and self.money[i]:
                    amountOfTimes = amount // self.values[i]
                    if amountOfTimes <= self.money[i]:
                        chipsAdded[i] += amountOfTimes
                        amount -= amountOfTimes * self.values[i]
                    else:
                        chipsAdded[i] = self.money[i]
                        amount -= self.money[i] * self.values[i]
            if amount == 0:
                self.lastbet = starting
                print(chipsAdded)
                self.minusChips(chipsAdded)
                self.endGo()
        else:
            self.endGo()



    def minusChips(self, arrayChips):
        for index in range(len(self.money)):
            self.money[index] -= arrayChips[index]


    def disconnect(self):
        self.stopping.set()
        self.recieveData()
        self.send('1[Disconnect]')
        # self.client.close()



if __name__ == "__main__":
    c = Client(username='Bartholomew Montgomery Clyde')
    c.money = [10, 5, 3, 2, 1]
```

```
    c.bet(65)
```

server.py

```
import socket, os, json, threading, pygame, base64, assetloader, time,
math, queue

packetID = {
    '0' : 'Connect',
    '1' : 'Disconnect',
    '2' : 'Player Data',
    '3' : 'New Game',
    '4' : 'Picture',
    '5' : 'Id',
    '6' : 'Loaded',
    '7' : 'Message',
    '8' : 'Start Game',
    '9' : 'Turn',
    '10' : 'Bet',
    '11' : 'Fold',
    '1000' : 'Ping',
}


#change port back to 6778

class Client():
    def __init__(self, username, pfp='', main=None):
        self.username = username
        self.pfp = pfp
        self.main = main

        self.client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.client.settimeout(1)
        self.connection = None

        self.turn = False
        self.betting = 0
        self.money = [0, 0, 0, 0, 0]
        self.values = [1, 2, 4, 8, 16]
        self.totalMoney = 0
        self.currentPlayer = None
        self.id = None
        self.networkQueue = queue.Queue()
        self.folded = False

        self.stopping = threading.Event()
        self.timeDebounce = 0


    def serialiseClient(self):
        data = {
            'Name' : self.username,
        }

        return json.dumps(data)

    def sendingImages(self, picture):
        try:
            with open('./User Images/' + self.pfp, 'rb') as f:
                convertedImage = base64.b64encode(f.read())
                self.client.sendall((convertedImage + b'\x1e'))
        except:
            with open(assetloader.imagePaths['no image'], 'rb') as f:
```

```
                convertedImage = base64.b64encode(f.read())
                self.client.sendall((convertedImage + b'\x1e'))

    def connect(self, serverIp, port):
        self.serverIp = serverIp
        self.port = int(port != ''  and port or 27015)
        try:
            self.client.connect((self.serverIp, self.port))
            threading.Thread(target=self.clientLoop, args=(),
daemon=True).start()
            return True
        except Exception as e:
            print("Connection error:", e)
            return False

    def send(self, data):
        try:
            self.client.sendall((data + '\x1e').encode())
        except Exception as e:
            print('Sending error:', e)

    def recieveData(self):
        data = b''
        while not self.stopping.is_set():
            try:
                recv = self.client.recv(4096)
                data += recv
                if not recv:
                    break
                elif recv[-1:] == b'\x1e':
                    data = data[:-1]
                    break
            except Exception as e:
                break
            except:
                break
        packetId = ''
        while len(data) > 0 and chr(data[0]).isnumeric():
            packetId += chr(data[0])
            data = data[1:]
        if packetId == '':
            return '1000', None
        return packetId, data

    def clientLoop(self):
        self.send(self.serialiseClient())
        self.sendingImages(self.pfp)
        while not self.stopping.is_set():
            packetId, data = self.recieveData()
            match packetID[packetId]:
                case "Connect":
                    pass
                case "Disconnect":
                    pass
                case "Id":
                    data = json.loads(data.decode())
                    self.id = data[0]
                case "Player Data":
                    print(data)
                    data = json.loads(data.decode())
                    self.main.game.players = data['Players']
```

```
                    self.turn = data['Current Player'] and (data['Current
Player']['Id'] == self.main.client.id)

                    if
len(self.main.game.mainPlayerHUD.playerDisplays.keys()) > 0:
                        for idx, val in enumerate(data['Players']):
                            if (val['Id'] != self.main.client.id):

self.main.game.mainPlayerHUD.playerDisplays[val['Id']].money = val['Money']

self.main.game.mainPlayerHUD.playerDisplays[val['Id']].action = val['Last
Action']

self.main.game.mainPlayerHUD.playerDisplays[val['Id']].username =
val['Name']

                    self.main.game.mainPlayerHUD.hand = [
                        self.main.Card(self.main, data['Hand'][0]['suit'],
data['Hand'][0]['value']),
                        self.main.Card(self.main, data['Hand'][1]['suit'],
data['Hand'][1]['value'])
                    ]
                    self.main.game.communityCards = []
                    for card in data['Community Cards']:

self.main.game.communityCards.append(self.main.Card(self.main,
card['suit'], card['value']))
                    self.main.game.mainPlayerHUD.drawHand()

                    if data['Current Player']:
                        self.currentPlayer = data['Current Player']['Name']

                    print(data)
                    self.main.client.money = data['Money']
                    self.main.game.mainPlayerHUD.role = data['Role']
                    self.main.game.pot = data['Pot']
                    self.main.game.lastbet = data['Last Bet']
                case "New Game":
                    for display in
self.main.game.mainPlayerHUD.playerDisplays.values():
                        display.remove2()
                    self.main.game.mainPlayerHUD.playerDisplays = {}
                    self.main.game.mainPlayerHUD.drawPlayers()
                    for clientId, playerInfo in
self.main.game.mainPlayerHUD.playerDisplays.items():
                        if os.path.exists('./pfp/' +  clientId + '.png'):

playerInfo.profilePicture.update(pygame.image.load('./pfp/' + clientId +
'.png'))
                    self.networkQueue.put(b'6[]\x1e')
                case "Picture":
                    data = data[1:]
                    clientId = ''
                    while len(data) > 0 and chr(data[0]).isnumeric():
                        clientId += chr(data[0])
                        data = data[1:]
                    data = data[1:]
                    picture = base64.b64decode(data + b'==')
                    with open('./pfp/' + clientId + '.png', 'wb') as f:
                        f.write(picture)
```

```
                    for cid, playerInfo in
self.main.game.mainPlayerHUD.playerDisplays.items():
                        if os.path.exists('./pfp/' +  clientId + '.png')
and cid == clientId:

playerInfo.profilePicture.update(pygame.image.load('./pfp/' + clientId +
'.png'))
                case "Message":
                    data = str(data.decode('utf-8'))
                    print(data)
                    if len(self.main.game.mainPlayerHUD.chatMessages) <= 4:

self.main.game.mainPlayerHUD.chatMessages.append(data[0])
                    else:
                        self.main.game.mainPlayerHUD.chatMessages.pop(0)

self.main.game.mainPlayerHUD.chatMessages.append(data[0])
                case "Turn":
                    self.main.client.turn = True
                    pygame.mixer.Sound("Assets/Sound/Ping.mp3").play()
                    self.totalMoney = self.calculateMoney()
                    self.betting = self.main.game.lastbet
                case "Start Game":
                    data = json.loads(data)
                    self.main.game.gameStart = data['Start Time']
                case _:
                    pass
            if not self.stopping.is_set():
                if not self.networkQueue.empty():
                    self.client.sendall(self.networkQueue.get())
                else:
                    self.client.sendall(b'1000[]\x1e')

    def minAmount(self):
        for idx, val in enumerate(self.money):
            if val >= 1:
                return self.values[idx]


    def increaseBet(self):
        minAmount = self.minAmount()
        if time.time() - self.timeDebounce >= 0.1:
            self.timeDebounce = time.time()
            if not self.betting + 1 > self.totalMoney:
                self.betting += minAmount

    def decreaseBet(self):
        minAmount = self.minAmount()
        if time.time() - self.timeDebounce >= 0.1:
            self.timeDebounce = time.time()
            if not self.betting - 1 < 0 and not self.betting - 1 <
self.main.game.getLastBet():
                self.betting -= minAmount

    def fold(self):
        self.folded = True
        self.betting = 0
        self.endGo()

    def endGo(self):
        if self.folded:
```

```
            self.networkQueue.put(b'11[]\x1e')
        else:
            self.networkQueue.put(bytes(f'10[{self.betting}]\x1e', 'utf-
8'))
        self.turn = False
        self.betting = 0

    def calculateMoney(self):
        value = 1
        totalMoney = 0
        for i in self.money:
            amount = value * i
            totalMoney += amount
            value *= 2
        self.totalMoney = totalMoney
        return self.totalMoney

    def bet(self, amount):
        starting = amount
        chipsAdded = [0, 0, 0, 0, 0]
        print(amount)
        if not amount == 0:
            for i in range(4, -1, -1):
                print(amount, self.values[i])
                if amount >= self.values[i] and self.money[i]:
                    amountOfTimes = amount // self.values[i]
                    if amountOfTimes <= self.money[i]:
                        chipsAdded[i] += amountOfTimes
                        amount -= amountOfTimes * self.values[i]
                    else:
                        chipsAdded[i] = self.money[i]
                        amount -= self.money[i] * self.values[i]
            if amount == 0:
                self.lastbet = starting
                print(chipsAdded)
                self.minusChips(chipsAdded)
                self.endGo()
        else:
            self.endGo()



    def minusChips(self, arrayChips):
        for index in range(len(self.money)):
            self.money[index] -= arrayChips[index]


    def disconnect(self):
        self.stopping.set()
        self.recieveData()
        self.send('1[Disconnect]')
        # self.client.close()



if __name__ == "__main__":
    c = Client(username='Bartholomew Montgomery Clyde')
    c.money = [10, 5, 3, 2, 1]
```

```
    c.bet(65)
```