Contents

[1.0 Analysis 4](#_Toc96937593)

[1.1 Introduction 4](#_Toc96937594)

[1.11 Background 4](#_Toc96937595)

[1.12 Product Description 4](#_Toc96937596)

[1.2 Investigation 5](#_Toc96937597)

[1.21 Prospective Users 5](#_Toc96937598)

[1.22 Questionnaire 6](#_Toc96937599)

[1.23 Existing Systems 7](#_Toc96937600)

[1.24 Constraints 9](#_Toc96937601)

[1.25 Flow Diagram 10](#_Toc96937602)

[1.26 Prototypes 11](#_Toc96937603)

[1.3 Objectives 12](#_Toc96937604)

[1.4 Language/Platform 14](#_Toc96937605)

[1.41 Proposed Solution 15](#_Toc96937606)

[2.0 Design 16](#_Toc96937607)

[2.1 Libraries 16](#_Toc96937608)

[2.2 Description of Data Items (Data Types and Structures) 17](#_Toc96937609)

[2.21 - User Information stored inside the Database 17](#_Toc96937610)

[2.22 - Gameplay Data 17](#_Toc96937611)

[2.23 - IPSO Diagram 18](#_Toc96937612)

[2.3 Data Types and Structures 18](#_Toc96937613)

[2.4 – Database Design 19](#_Toc96937614)

[2.41 - Entity-Relationship Diagram 19](#_Toc96937615)

[2.42 Example Queries 19](#_Toc96937616)

[2.5 Class Diagram 21](#_Toc96937617)

[2.6 – UI Design / HCI Rationale 22](#_Toc96937618)

[2.61 -Login Screen 22](#_Toc96937619)

[2.62 - Register Screen 23](#_Toc96937620)

[2.63 - Main Menu 24](#_Toc96937621)

[2.64 – Game Mode Selection Screen 24](#_Toc96937622)

[24](#_Toc96937623)

[2.65 – Options Screen 25](#_Toc96937624)

[2.66 – Game Play Screen 25](#_Toc96937625)

[25](#_Toc96937626)

[25](#_Toc96937627)

[26](#_Toc96937628)

[26](#_Toc96937629)

[2.7 – Description Of Measures Planned For System Security 27](#_Toc96937630)

[2.8 – Pseudocode and Algorithms 28](#_Toc96937631)

[2.81 - Get Text 28](#_Toc96937632)

[2.82 - Change The Volume 28](#_Toc96937633)

[2.83 - Create The Deck 29](#_Toc96937634)

[2.84 - Check if the card placed down is a special card 29](#_Toc96937635)

[2.85 - When the player chooses to draw a card 30](#_Toc96937636)

[2.86 - Client Side 30](#_Toc96937637)

[2.9 – Test Strategy 31](#_Toc96937638)

[3.0 Technical Solution 32](#_Toc96937639)

[server.py file 32](#_Toc96937640)

[client.py file 35](#_Toc96937641)

[game.py file 38](#_Toc96937642)

[actions file 41](#_Toc96937643)

[deck.py file 43](#_Toc96937644)

[buttons.py 47](#_Toc96937645)

[interface.py file 48](#_Toc96937646)

[login.py 50](#_Toc96937647)

[menu.py file 52](#_Toc96937648)

[game\_screen.py 60](#_Toc96937649)

[4.0 – Testing 73](#_Toc96937650)

[5.0 Evaluation 76](#_Toc96937651)

[5.1 - Navigating through the menus 76](#_Toc96937652)

[5.2 - Multiple game modes and multiple games 76](#_Toc96937653)

[5.3- Reading the rules 76](#_Toc96937654)

[5.4 - Adjust the volume 77](#_Toc96937655)

[5.5 - Sending data back and forth 77](#_Toc96937656)

[5.6 - Shuffle the deck and deal 7 cards 77](#_Toc96937657)

[5.7 - Viewing your cards and the opponent’s cards 78](#_Toc96937658)

[5.8 - Placing a card down or drawing a card 78](#_Toc96937659)

[5.9 - Uno functionality 78](#_Toc96937660)

[5.10 - Other functionalities 79](#_Toc96937661)

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

# 1.0 Analysis

## 1.1 Introduction

### 1.11 Background

UNO is a card game created in 1971 by Merle Robbins. Since the game’s introduction, it has sold increasingly well and has become one of the most popular card games played around the world. Since it’s inexpensive to buy and easy to learn, Uno appeals to a wide age range and continues to be enjoyed by both children and adults.

Although it is popular, there is still a large number of people who are unaware of the rules and do not know how to play. Therefore, I will be creating my own version of an UNO Card Game to allow users to practice and learn the rules through playing the game repeatedly.

### 1.12 Product Description

My version of the game will stick to the original rules and there will be a selection of options such as two, three and four-player modes. The user has a choice of playing against another real player, or against an AI. Through the implementation of client-server architecture, other users will be able to connect to the game and play against you. Additionally, I will be making an AI player that you can play against or use to fill in the slots if there aren’t enough real players available.

Your wins and losses in each game mode will be recorded. Therefore, a login system will be imperative so that users must create an account and log in before playing the game. This is necessary so the system can store data for multiple users.

**Rules of UNO**

Each player is dealt 7 cards initially and the remaining cards are placed facedown to form a draw pile. The top card of the draw pile is placed down to form the discard pile.

The rules are fairly simple: players take turns matching a card in their hand with the current card in the discard pile, either by colour or number. For instance, if a Yellow 2 is placed in the discard pile, the next player must place any yellow card or any coloured 2 from their hand. If a player cannot match the card, they must draw from the draw pile.

In addition to the normal cards, there are 5 special cards:

* Draw 2 Card – When you play this card, the next person to play must draw 2 cards and their turn is skipped.
* Reverse Card – This card reverses the direction of play.
* Skip Card – This card causes the next person to lose their turn.
* Wild Card – Placing this card down allows you to select and change the colour being played to any colour.
* Draw 4 Wild Card – This card allows you to change the colour being played and requires the next player to pick 4 cards from the draw pile, and their turn is skipped. However, this can only be placed when you have no cards that match the colour of the card previously played.

The aim of the game is to be the first player to get rid of all of your cards. However, you must remember to say “UNO” before your second-to-last card touches the discard pile, or if a player notices and calls you out, you must draw 4 cards. After getting rid of your cards, you gain points for the cards remaining in other player’s hands. Number cards are worth their face value but Draw 2, Reverse, and Skip cards are worth 20 points, and Wild and Draw 4 Wild Cards are worth 50 points. The first player to reach 500 points is the final winner.

## 1.2 Investigation

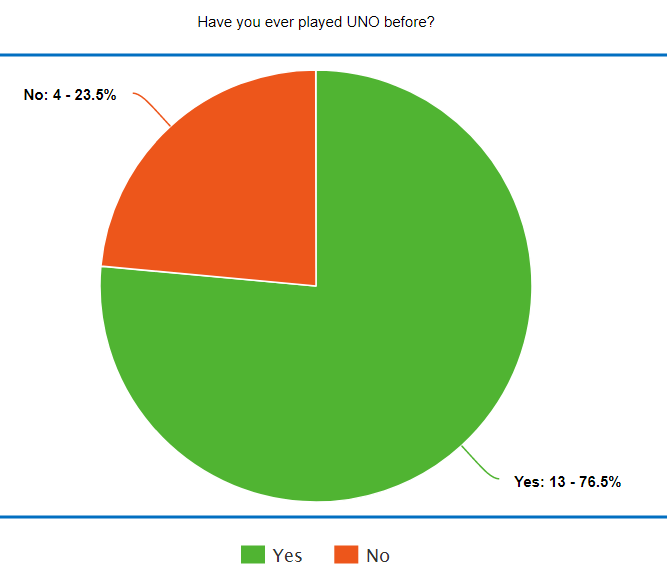
### 1.21 Prospective Users

As mentioned earlier, Uno is reasonably simple and hence it can be played by children and adults. However, I need to consider the fact that young children are unlikely to have the ability to perform actions such as creating an account and retrieving a verification code from their email.

Therefore, my target audience will be people aged around 8 or above and thus I have chosen my friends and family as my end users who will be playing my game. They are all computer literate so their skills shouldn’t limit the complexity of my game but I will attempt to keep things like navigating through options simple nonetheless.

### 1.22 Questionnaire

Through verbal feedback and the use of a poll I sent to friends I gained multiple responses. As seen on the chart, 76.6% of people have played UNO before, whereas 23.5% have not. This emphasises my earlier statement of UNO being an extremely popular card game but of course, a small minority have not played so I’ll need to make sure my game is easy to play and understand for first time players.



Furthermore, I asked the people who responded to inform me about any aspects or features of the game that they enjoyed and did not enjoy, or any possible improvements. The majority of people stated that they enjoyed how simple the game was and how easy it was to learn how to play. I hope to inherit this aspect into my game, making it user-friendly for all.  
One of my friends who have played the mobile version of UNO frequently in the past provided me with some useful knowledge of the cons of the game. For instance: the time limit, which can be frustrating. Players who do not perform any actions during their turn and run out of time will have a card picked out for them automatically. This means that many players can leave their devices unattended without having to manually pick a card, thus wasting the time of the other players. Although the time limit shortens over time, perhaps I can eliminate this from my game completely by skipping a player’s turn if they do not pick a card in time.

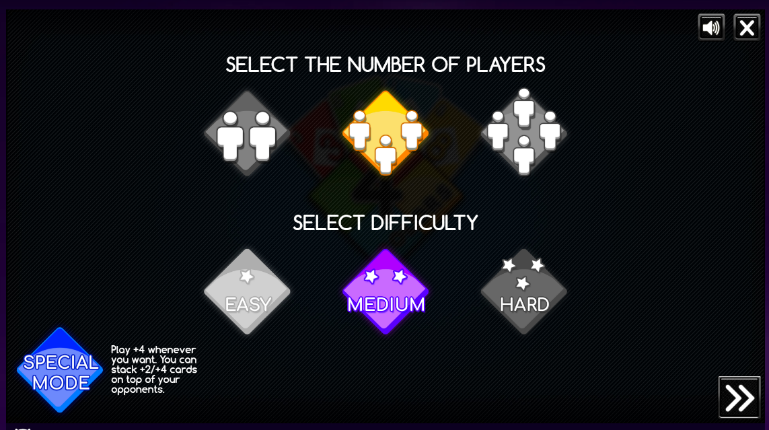
### 1.23 Existing Systems

I have looked at existing versions of Uno games to gain a deeper insight into what my game will look like and to see if I can improve on existing functionalities.

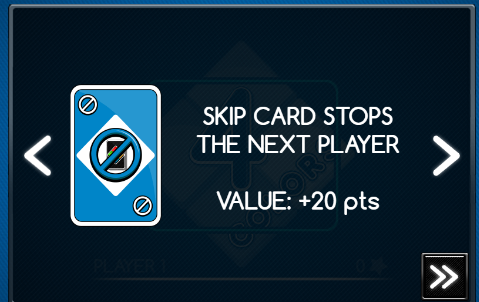


This is an online version of UNO, which is accessible via any web browser using the link <https://www.crazygames.com/game/uno-online>.

This version of the game had no login system meaning that players could not keep track of the number of wins and losses that they had.



After clicking play, you are directed to a screen that allows you to choose the number of players and the difficulty. There is also a special game mode where the rules are changed slightly. In this game mode, +4 can be played at any time and +2/+4 cards can be stacked. Although I will not include an option for the difficulty, I will allow the user to select the number of players.



Following this, the screen changes to display the rules. This is a useful feature because it allows the users to attain knowledge of the rules and the effect of the special cards before playing. However, new players may forget the effects of the special cards during the game, so I will implement a button that opens up a webpage of the rules. Users can look at this webpage as they play to aid them.



The game functions extremely well, with no errors regarding the actual gameplay. Users can select a card to place down, and the AI does the same when it is their turn.

There is background music that can be muted, as well as sound effects when a player uses a special card.

Once a player places their final card down, their score is calculated using the value of their opposing player’s cards and subsequently, the game restarts. This continues until a player reaches 250 points, in contrast to the usual 500 points. However, my game will not restart after it ends as an excessive number of rounds may need to be played.



I have noticed some possible improvements that I can implement into my game. For example, this system doesn’t display the current direction of play so it can be confusing as to whose turn it currently is if the reverse card is used continually. Also, there is no time limit during your turn, so perhaps I could include one to develop a sense of urgency and improve one’s decision-making skills.

This is the official Uno mobile game.

Likewise, there are no major issues with the gameplay. Although I noticed that the opponent’s cards aren’t placed down flat making it hard to determine how many cards your opponent has left, such as with the user on the right. The background of the game is also bright with various colours and objects but I find this slightly distracting so I am likely to use a simpler background.

****

### 1.24 Constraints

**Hardware Constraints**

My game will be able to run on Microsoft Windows Devices such as PCs and laptops but not mobile devices. I’m using a Windows 10 laptop so I’m uncertain about the performance of my game on a MacBook device, but these devices should be able to run my game as Python is available on MacOS, as well as the Pygame module.

**Software Constraints**

Devices that run on iOS and Android will be unable to run this game on their devices. Users will require an internet connection to download my game, along with the necessary files, before they can play it. My game will also make use of a client server network to allow another user to join the game, thus requiring an internet connection. Furthermore, Python and a set of python modules named Pygame will need to be downloaded beforehand, in order to run my code.

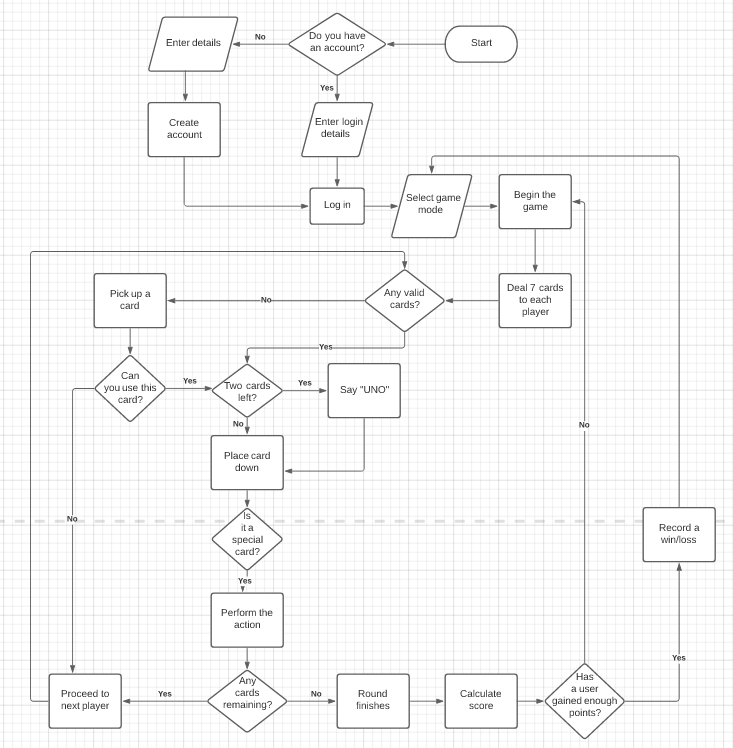
**User’s Knowledge of information technology**

The user will require a basic understanding of using computers since various actions need to be performed before they can access the game. For example, they will need to log into their email to obtain their verification code. Python also need to be installed beforehand, and the game itself needs to be downloaded from GitHub.

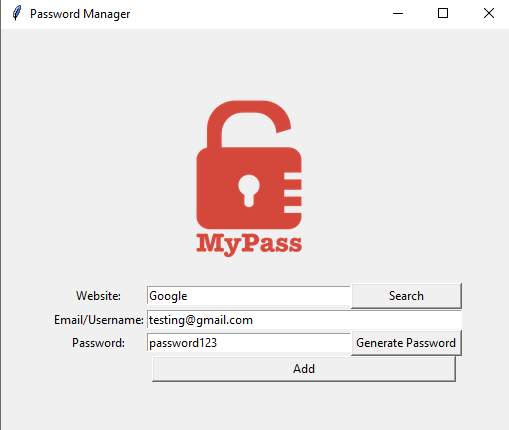
However, the system itself will be user-friendly and the necessary actions such as entering their login details and navigating through the various options should be fairly simple to complete.

### 1.25 Flow Diagram

This diagram represents the basic processes and decisions that will be made when logging in and selecting the classic game mode.



### 1.26 Prototypes

****

To develop a prototype for my login screen, I have created a graphical user interface using Tkinter in Python. I made a password manager that lets users create accounts for websites and store these details inside a file. Users can use their own passwords, or the program can generate a password for you, as seen in the images. I experimented by storing the details in various formats, such as JSON, CSV and text files.

However, this is simply a prototype I used to develop the skills needed to produce a login system and some components of this prototype will not be used in my game. Such as the Search function that allows you to search for your login details for a specific website.

In the final version of my game, there must be additional options such as a button to log in and an entry box to enter your verification code.

## 1.3 Objectives

* New users must create an account to use the system.
* The new user should enter their new username, password and email
* The username and password must be unique and not already used
* The user should be able to generate a password if needed
* The user must answer a question to verify they are human
* The user should be able to log in to an existing account.
* The user must enter their username, password and email.
* The system must notify the user if fields are missing.
* The system must store the login details inside a database and allow new users to add their login details when creating an account.
* The system should track the number of wins and losses in each game mode by updating the database
* The system should display this information to the user when they are selecting game modes
* The tables in the database should be interlinked.
* For example, one table containing the user details will link to a table containing the wins and losses. This allows you to extract the data from the tables easily.
* The user should be able to navigate through the menu using the arrow keys.
  + The screen should change to the options screen or game mode screen if selected.
  + The user must be able to pick between two/three/four-player mode.
* The user should be able to read the rules when needed, by clicking on a button and being redirected to a webpage.
* There should be an option to adjust the volume or completely mute the sound.
* Multiple games should be able to run simultaneously
  + The games should not interfere with each other and data should not be sent to the wrong game.
* The client and server should be able to send data back and forth between each other.
  + The server should receive the action from the client and then execute the action.
  + All clients must receive the updated game after a player has made a move so that all players can see which card was just placed down.
* The system should shuffle the deck and deal 7 cards to each player.
* The remaining cards should form a draw pile and the top card of the draw pile should be placed down when starting the game
* The user should see all of his cards and how many cards his opponents have
* The user should be able to select one of their cards to place down.
* They will be notified if they choose a card that cannot be used.
* If they have no valid cards, they should be able to draw a card from the deck. If this card is valid, the player must be asked whether they want to place that card down.
* If they chose a wild card, they must be given the option to select a colour to change to.
* A button labelled “Uno” will be displayed on the screen during gameplay
* If the user forgets to click it before placing his second-to-last card down, the system should notify them and the user will draw four cards.
* The system should display the direction of play during gameplay so the user knows whose turn it is next.
* The system should explain the effect of the wildcard placed down.
* For instance, if Player 1 uses a Skip Card, the system will display a message, explaining that Player 2’s turn will be skipped.
* The round should finish once a player places down their final card.
* The system should direct the user back to the game mode selection screen once the game has finished.

## 1.4 Language/Platform

There are a range of options for me to choose from regarding the platform and libraries used. Each have their own advantages and disadvantages.

Python

* **Tkinter** – This will enable me to create a GUI easily through the use of labels, buttons and more, plus I have a basic understanding of using it.
* **Pygame** – This library is designed for creating games, and will allow me to control the movements of the cards more easily. Although, it will be exceedingly more difficult to create a GUI and widgets such as buttons, which I may need to rely on heavily.
* There are many other modules for me to choose from to aid me in creating the game, such as PyQT or PySimpleGUI.

C#

* **Windows Form** – This allows me to drag and drop forms onto the screen to create a GUI easily. A variety of controls can be added to the forms, such as buttons.

### 1.41 Proposed Solution

My UNO card game will be created in Python using the Pygame module as opposed to Tkinter as it will make my solution considerably more complex and provide extra functionalities that Tkinter is incapable of. For instance, the entry boxes and buttons used will be created myself by taking into account attributes such as the x and y coordinates of the mouse’s current position. In regards to the programming language used to implement the game, I have chosen Python because I have a considerable amount of experience in contrast to C#.

The game will make use of sockets so another player can join the game and play against you. All inputs by clients will be processed by the server and the server will send data back to the players so they receive updates in real-time. The delay between moves should be minimal to allow for smoother gameplay.

The server will be hosted on my laptop and clients will be able to connect to my server using my public IP address that I will provide them with. However, if I run into any issues I may decide to use an alternative and host the server online using a service called Cloud Linode, who offer $100 free credit which would last for the development of the game.

The game will be connected to a database on the server and I will use SQL queries to create the tables in the database, store data and retrieve it. For example, a table containing their login details and another table that records their wins and losses will be created. Their login details will need to be retrieved and compared against their inputted details to verify the players before allowing access into the game. Their wins and losses will be retrieved in order to display this information to the user when selecting game modes.

# 2.0 Design

## 2.1 Libraries

**pygame** – This library will be used to create most of the features in my GUI, including the buttons. I will also need this module to display my card images onto the screen, which I will be downloading from the internet to ensure they are of the same size and font.

**socket** – The socket library is essential for creating servers and clients and I will be using this to bind IP addresses to ports, to allow the clients to connect to the server.

**pickle –** This module implements binary protocols which enables the clients and server to send and receive data. The pickle module is useful for converting any type of data, including strings and objects, into a byte stream which can be sent. When the data is received, it is converted back into the original data.

**threading –** This module allows me to use multithreading so that multiple games can run at the same time. Every game will run in its own thread so that they do not interfere with each other.

**time –** My code will make use of the time.sleep() function from this module to add short delays in between sending and retrieving data to synchronise the clients. Without delays, I am likely to run into issues as the server may send data to the client before it is ready to receive it.

## 2.2 Description of Data Items (Data Types and Structures)

### 2.21 - User Information stored inside the Database

This is the information entered by the user when logging in or registering. I will validate the user input to ensure that they do not crash or break my program. These are examples of the data inputted by the user.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Field Name** | **Purpose** | **Field Type** | **Field Size** | **Example** | **Validation** |
| Username | Stores the player’s username | String | 15 | Bob03 | No special characters.  Must be between 3 and 15 characters long. |
| Password | Stores the encrypted version of the user’s password | String | 30 | Xykgfd23 | Not null.  Must be under 30 characters in length. |
| Email | Store’s the user’s email | String | 30 | bob@gmail.com | Not null Must be in the correct format. |

### 2.22 - Gameplay Data

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field Name** | **Purpose** | **Field Type** | **Field Size** | **Example** |
| Win | Stores the number of times the user has won in that game mode | Integer | 5 | 15 |
| Loss | Stores the number of times the user has lost in that game mode | Integer | 5 | 23 |
| Number Of Games Played | Stores the number of times the user has played in that game mode | Integer | 5 | 8 |

### 2.23 - IPSO Diagram

|  |  |
| --- | --- |
| Inputs  -User data (Username, passwords…)  -Mouse clicks  -Keyboard Presses | Process  -Validation check  -Encrypt password using Caesar cipher  -Create SQL queries to insert user data  -Create SQL query to retrieve/update the number of wins, losses and games played.  -Get position of mouse and compare it to the position of the buttons  - Check which key was pressed and perform the necessary action |
| Outputs  -Display the user stats (Wins, losses, number of games played, game mode)  -Output the cards onto screen  -Update the new position of the cursor onto screen | Storage  -User’s personal data  -Number of wins, losses and games played |

## 2.3 Data Types and Structures

Throughout my program I will be using various data types and data structures.

* Boolean – This will be used throughout the code as variables will often need to be checked if they are set to True or False so that another process can occur.
* Floats – To store data such as the sound volume.
* Tuples – This will be used often to store RBG values to specify the colours, as well as the coordinates of various objects.
* Strings – The user ID’s will be converted from an integer to a string so that it can be displayed onto the screen using pygame. String data such as messages will also be sent back and forth between the clients and servers occasionally.

Data Structures:

* Dictionaries - They will be essential on my server when creating multiple games. Every game has its own ID number and these will be linked together via key-value pairs.
* Lists - Objects such as buttons may be stored in lists to allow me to loop through the list and find the required button. Player decks will be lists since any card in the deck can be selected and these will need to be accessed via their indices.
* Stack – The main deck will be a stack since the top cards will need to be popped off when dealing the cards or when a player draws a card. It is unlikely that the players will use all cards in the deck so adding them to the back of the deck may not be necessary, hence a queue may not be needed.

## 2.4 – Database Design

### 2.41 - Entity-Relationship Diagram

My database will be a simple one-to-one database as very little data will be stored. The Player table will contain the username, password and email of every single player that has registered. Each game mode has its own table which also contains the Username, along with the number of wins and losses that user has in that game mode. The username is a primary key in every table which allows me to retrieve the data easily.

### 2.42 Example Queries

**Inserting a new user into the database:**

INSERT INTO Players  
VALUES (“John03”, “qwerty123”, “john@gmail.com”)

**Incrementing the number of wins in a table:**

UPDATE Two\_P  
SET Wins = Wins + 1   
WHERE Username == “John03”

**Incrementing the number of losses in a table:**

UPDATE Three\_P  
SET Losses = Losses + 1   
WHERE Username == “John03”

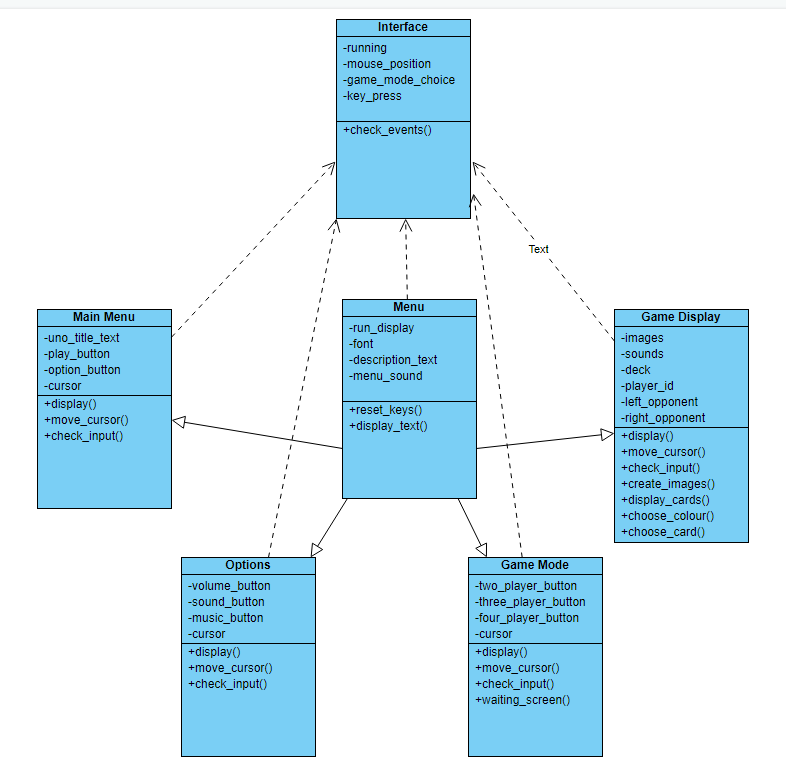
**Incrementing number of games played:**

UPDATE Players  
SET Games\_Played = Games\_Played + 1   
WHERE Username == “John03”

## 2.5 Class Diagram

This is a diagram of my Menu, Options, Game Mode and Game Display screens. As indicated by the normal arrows, these four classes will all inherit from one Menu class. This grants the four classes access to the attributes and methods in the Menu class. A benefit of this would be that the two methods inside the Menu class can be called from anywhere inside the other classes, without it needing to be rewritten. Every screen having its own class also makes debugging simpler and helps me break down the task. The font will be inherited to keep it consistent throughout all screens.

Every class has their own display, move\_cursor and check\_input method because the classes each contain different attributes and buttons, so different objects would be displayed on the screen. This is the reason that I chose not to have each class inherit the three methods from the Menu class.

Furthermore, the dashed arrows show that all 5 classes are dependent on the Interface class. This is due to the class objects being created and contained inside of the Interface class. All 5 classes will take the Interface as a parameter, enabling each screen to check for key presses on the same Interface and to move the cursor and display the necessary items onto the same screen. Any changes to the Interface object will affect all the other classes, thus they are all dependent on the Interface.

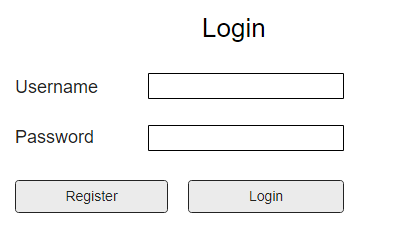
## 2.6 – UI Design / HCI Rationale

My aim is to optimise the ease of navigation to allow all users to select the menus and options easily, so I will ensure that all elements are positioned and sized appropriately. As the cards will be of various colours I have chosen to use a darker colour scheme to allow the cards to stand out during gameplay and to minimise confusion. The colour of the text must not overlap with the background colour either.

For my login and register screens, there will be several input boxes/buttons to input data such as their usernames and passwords. These elements will light up when the user clicks on them to indicate that they have been selected.

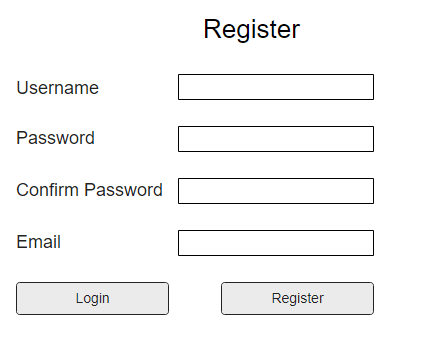
### 2.61 -Login Screen

The login screen is where the user can login to the system. There will be two input boxes to allow the user to enter their username and password, as well as two buttons labelled “Login” and “Register”. If the user has entered their details they can click the login button where they will be sent to the main menu. However, if the user wishes to register then they can click the register button to access the register screen.

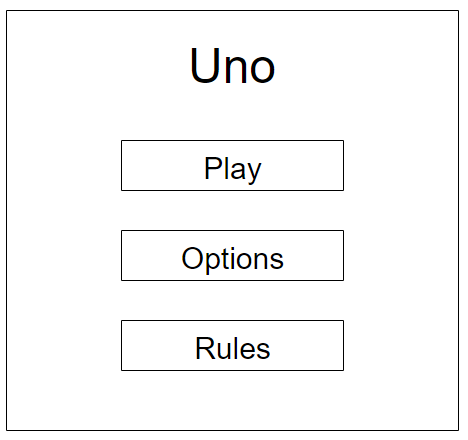


### 2.62 - Register Screen

The register screen allows the user to register a new account. It will have four input boxes for the username, password, email and an additional one to confirm your password. The text in the two boxes must match or an error will be displayed on the screen. If the user inputs are valid then a request will be sent to the server, after clicking register, and the account will be registered with the system. In addition to the register button, the login button allows you to return to the login screen in case you accidently clicked on register.



### 2.63 - Main Menu



The main menu screen will be displayed once the user has successfully logged in. It will consist of multiple buttons which allow the user to navigate through different screens such as the Game Mode selection screen and the Options screen.

The user will be able to highlight a button using the arrow keys, and pressing Enter will select that button. To return to the previous screen, the user can press the Back button. I will add some text at the bottom of the screen to tell the user about this.

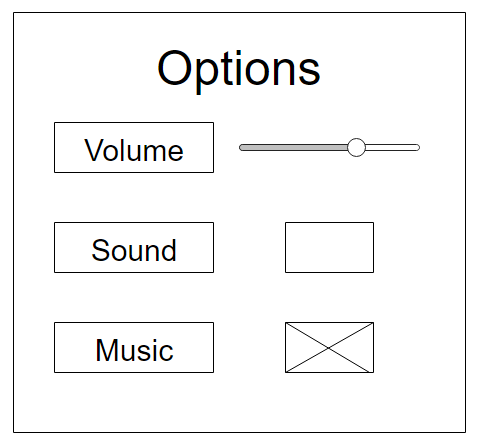
Selecting the Rules button will open up a website that explains the rules for new users.

### 2.64 – Game Mode Selection Screen

## 

This screen allows the user to select the game mode that they would like to play. This information will be sent to the server to be processed and the game will begin once enough players have joined.

### 2.65 – Options Screen



The purpose of this screen is to let the user adjust the volume for the sounds and music as it may be too loud/quiet for the user. They may also completely mute the sound and music if they wish to by selecting the corresponding box and pressing enter. A cross will be displayed inside the box if they have chosen to disable it.

### 2.66 – Game Play Screen

## 

## 

This is how I intend the two player mode to look. There will be no cards on the right and left sides compared to the other game modes since there are only two players.

The card at the top of the discard pile will be placed in the centre of the screen and a draw button will be placed directly under it, to give the user the choice of drawing a card.

Additionally, the player numbers will be placed alongside the player’s cards to differentiate between players.

## 

The three player mode will look slightly different as there is no player at the top of the screen. The opponent’s cards will be placed on either side of the screen.

To select a card, the users will use their arrow keys and press enter to select the card. I will add a cursor around the cards to let you see which one you are selecting.

## 

This is the layout of the four player mode. The opponent’s cards will be placed at the top, left and right sides of the window.

As there are more cards on the screen, any text that I display during the game will likely be adjusted slightly. For instance, the text that I will include to inform the players of the current player.

When you place down a card, the position of your remaining cards will shift along and remain centered to ensure they never display outside of the window.

### 2.7 – Description Of Measures Planned For System Security

My project makes use of sockets which allows clients to connect to my server over the internet. My server is hosted on my laptop at home, so I need to distribute my public IP address to anyone who wishes to play my game. This is a risk since someone who gains access to my public IP address may want to use it for malicious purposes. For instance, a hacker may attempt to brute-force a connection by trying all the possible ports associated with my IP address to gain access to my device. Or, they may attempt a DDoS attack to flood my device with traffic.

To prevent this from happening, I will only be distributing my public IP address to people that I know and trust. This is not a major issue as my game was designed for friends and family. Furthermore, I will minimise the chances of any attacks by running the server only when someone requests to play my game.

If someone gains access to my server they will be able to view data such as usernames, passwords and emails. To make this more secure, I can use hashing algorithms and encryption such as Caesar Cypher when storing the data.

To prevent SQL injection, user inputs on the login screen will also be validated. This will prevent people from accessing the database without authorisation and prevent my program from crashing. If I have enough time, I may attempt to use parameterized queries too.

## 2.8 – Pseudocode and Algorithms

### 2.81 - Get Text

This algorithm obtains the unicode of the key pressed by the user and adds it to a text variable so that it can stored and displayed onto the screen. This is vital for the login and register screen so the user can input their data.

Get\_text():  
 if user pressed a key:  
 for button in buttonlist: # Loops through the buttons on the screen  
 if button is active: # Finds the button that the user selected  
 if key = backspace:  
 Remove the character from the end of the text  
 else:  
 text += key.unicode

### 2.82 - Change The Volume

This is how the player will be able to adjust the volume on the options screen

Change\_volume():  
 if user pressed left\_arrow\_key:  
 move slider to left  
 decrease volume  
 else if user pressed right\_arrow\_key:  
 move slider to right  
 decrease increase  
 else if user pressed the up or down arrow key:  
 move cursor up or down  
 else if user pressed Enter and sound button is active:  
 mute or enable the sound  
 else if user pressed Enter and music button is active:  
 mute or enable the music

### 2.83 - Create The Deck

This will create the card objects by assigning them a colour and a value, such as red and 7.

Create\_deck():  
 colours = list containing red, blue, green and yellow

For number in range(2): # There are 2 of the same cards so it loops twice  
 for number in range(13):  
 # 1-9 will be the normal numbers and 10-12 are the special cards

for colour in colours: # create a card of this number for each colour  
 if number = 10:  
 create draw 2 card and give it the current colour  
 else if number = 11:  
 create reverse card and give it the current colour  
 else if number = 10:  
 create skip card and give it the current colour  
 else: # Any normal numbered card  
 create the card and give it the current colour  
 next colour

for number in range(4):  
 create wild card and wild 4 card

### 2.84 - Check if the card placed down is a special card

This function will perform the necessary action if a special card was placed down. For instance, the next player will be skipped if a skipped card is placed down.

Check\_card():

If card at top of discard pile is a draw 2 card:  
 the next player draws 2 cards  
 skip the next player  
else if card at top of discard pile is a skip card:  
 skip the next player  
else if card at top of discard pile is a reverse card:  
 skip the next player  
 change the direction  
else if card at top of discard pile is a wild 4 card:  
 the next player draws 4 cards  
 skip the next player  
else: # It is a normal card  
 move onto next player

### 2.85 - When the player chooses to draw a card

A DrawCard object will be sent to the server by the client and the server will execute the method inside the object which will deal a card to the player and then checks if that card can be placed down. If it is valid, the server will send data back to the client to inform the client that they should decide whether they want to place the drawn card down.

Similarly, if they want to place a card down, the action method would simply place their card down, by removing it from their deck and adding it to the discard pile. Then it would move onto the next turn.

Class DrawCard:

action():  
 give the player a card  
 if the card’s colour or value matches the card at the top of the discard pile:  
 send data to the client to ask the user if they want to place it down  
 else: # Doesn’t match  
 move onto the next player

2.86 - Client SideWhen a person runs the client code they will connect to the server and then a window will pop up on their screen allowing them to navigate through the Menus and choose a game mode. Once they chose a game mode, they will be asked to wait until enough players have joined so that the game can begin. This is a simplified version of the processes.

Connect to the server via it’s IP address and port

While loop:  
 display menu

if a game mode was selected:  
 send the selected game mode to the server  
 display the waiting screen  
 if data is received from the server: # This means the game has begun   
 stop displaying the waiting screen

else:   
 continue displaying the menu

While loop:  
 receive your player ID from the server  
 receive the game from the server

if your player ID = the current player’s ID: # It is your turn  
 display the game screen   
 action = choose or draw a card  
 send action to server

else: # Not your turn  
 display the game screen

wait to receive data from server  
 continue the loop to receive the updated game

### 2.9 – Test Strategy

I will be carrying out a variety of functional testing to ensure that the algorithms and functions in my code work correctly. For instance, I will test every button on the menu and in game to ensure that they work and I will check if text is being displayed correctly when necessary. I will also attempt to place down every type of special card to ensure that the actions are performed correctly.

# 3.0 Technical Solution

## server.py file

**import** socket  
**import** threading  
**import** pickle  
**from** game **import** Uno  
**from** actions **import** Response  
**import** time  
  
server = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
ip = socket.gethostbyname(socket.gethostname()) *# my IPv4 Address*port = 5555  
  
ADDR = (ip, port)  
  
print(**"Server is starting"**)  
**try**:  
 server.bind(ADDR)  
**except** socket.error **as** e:  
 print(e)  
  
server.listen() *# lets multiple clients connect*print(**"Waiting for a connection\n"**)  
  
  
**def** close\_connection(connections):  
 *""" Closes connection for all players """* **for** conn **in** connections:  
 conn.close() *# Closes connection for all players* time.sleep(0.1)  
  
  
**def** play\_game(game, conn\_list):  
 **for** num **in** range(0, len(conn\_list)): *# To get the connections of each client* conn\_list[num].sendall(pickle.dumps(**"Started"**)) *# To let all players know that the game has begun* time.sleep(0.05) *# A delay is needed as the client would not be ready to receive the data yet (error occurs)* conn\_list[num].sendall(pickle.dumps(num)) *# To let the player know which number they are* time.sleep(0.1) *# Short delay stops data being sent to wrong client* running = **True** confirming = **False** data = **None  
  
 while** running:  
 **if not** confirming: *# If it's waiting for the player to confirm their choice it doesn't send anything* **for** conn **in** conn\_list:  
 *# Every player receives a request from the server but only if it's their turn, they will send data back* response = Response(game, **"Choose"**)  
 conn.sendall(pickle.dumps(response)) *# Telling the clients to select or draw a card  
 # Send a response object to the client which contains the game and an additional message (payload)  
 # On the client side, the player whose turn it is will send their action back to the server* time.sleep(0.1)  
  
 **try**: *# Could add a timer here* data = pickle.loads(conn\_list[game.turn].recv(2048\*3)) *# Receiving the action from the current player* **except** Exception **as** e:  
 print(e)  
 print(**"A player has left so the game will stop."**)  
 close\_connection(conn\_list)  
 running = **False  
  
 else**:  
 confirming = **False** *# Reset it in case a player was asked to confirm their choice before* response = data.execute(game) *# Calls the execute method of the action and the updated game is returned  
 # E.g Executing the PlaceCard action removes the card from the player's deck and adds it to the discard pile* **if** response.payload == **"Executed"**: *# If they placed a card down  
 # Checks which card was selected and performs the necessary action:* response.game.compare\_card() *# e.g A Draw 2 card skips the next player and deals them 2 cards* **elif** response.payload == **"Confirm"**:  
 *# If the player drew a card they'll be asked if they want to place it down* confirming = **True** *# So the server doesn't ask the current player to choose another card  
 # until they've confirmed if they want to place the card they drew down or not* game = response.game *# Update the game so the updated version is sent to all players* **if not** confirming:  
 **for** conn **in** conn\_list:  
 conn.sendall(pickle.dumps(response.payload))  
 *# To synchronise all clients so they receive the game below at the same time ^* time.sleep(0.05) *# Delays ensure the clients are ready to receive data before it is sent* conn.sendall(pickle.dumps(response)) *# Send the game along with a message back to all clients* time.sleep(0.1)  
  
 **else**:  
 *# When the current player is confirming, it only sends data to them so that the other players will wait* conn\_list[game.turn].sendall(pickle.dumps(response.payload))  
 conn\_list[game.turn].sendall(pickle.dumps(response))  
 time.sleep(0.1)  
  
 **if** game.finished: *# Player placed down their final card* close\_connection(conn\_list)  
 running = **False**games = {} *# Dictionary to store the game id along with the corresponding game object*conn\_list = []  
  
game\_id = 1  
player\_id = 0  
  
games[game\_id] = Uno() *# Creating a game for the first group of players*game\_mode\_dict = {2: [],  
 3: [],  
 4: []} *# Every game mode has their own list to store the player's connection***while True**:  
 connection, addr = server.accept()  
 print(**f"\n{**addr = **}"**)  
 print(**f"{**game\_id = **}"**)  
  
 game\_mode = pickle.loads(connection.recv(2048\*2)) *# Receive every client's game mode choice* print(**f"{**game\_mode = **}"**)  
  
 *# Adds the connection of the player into a list depending on their choice* game\_mode\_dict[game\_mode].append(connection)  
  
 **for** game\_mode **in** game\_mode\_dict: *# To begin the game when enough players have chosen that game mode* **if** len(game\_mode\_dict[game\_mode]) == game\_mode: *# E.g If 3 players have chosen Three Player mode* **for** i **in** range(0, len(game\_mode\_dict[game\_mode])):  
 games[game\_id].add\_player(i) *# Create the players inside the game* games[game\_id].start\_game(len(game\_mode\_dict[game\_mode]))  
 *# Tells the game object which game mode was selected and deals cards to each player* thread = threading.Thread(target=play\_game, args=(games[game\_id], game\_mode\_dict[game\_mode]))  
 thread.start() *# Creates and starts a new thread  
 # Each game will play in their own thread so multiple games can play at the same time* game\_mode\_dict[game\_mode] = [] *# Clear the list so new players can also play that game mode* game\_id += 1 *# So the next game created has a different game\_id* games[game\_id] = Uno() *# Create a new game for the next players*

## client.py file

**import** socket  
**import** pickle  
**from** interface **import** Interface  
  
  
**class** Network:  
 **def** \_\_init\_\_(self):  
 self.client = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 self.ip = socket.gethostbyname(socket.gethostname())  
 *#self.ip = "178.79.144.126"  
 #self.ip = "80.4.69.71"* self.port = 5555  
  
 self.addr = (self.ip, self.port)  
 self.client.connect(self.addr)  
  
 **def** send(self, data):  
 **try**:  
 self.client.sendall(pickle.dumps(data)) *# Sends data to the server* **except** socket.error **as** e:  
 print(e)  
  
 **def** receive(self):  
 **return** pickle.loads(self.client.recv(2048\*3))  
 *# pickle.loads to de-serialize the data which is in bytes format, and return the reconstructed object***def** main():  
 interface = Interface()  
  
 sent = **False  
 while** interface.running:  
 interface.current\_screen.display() *# Prompts client to login and allows them to navigate through the menus* **if not** sent **and** interface.game\_mode\_choice != 0: *# If they chose a game mode & they haven't sent it to server* net = Network() *# Connects the client to the server after they select a game mode* net.send(interface.game\_mode\_choice)  
 sent = **True** *# So the client only sends it once* net.client.setblocking(**False**) *# Set socket into non-blocking mode so the loop doesn't pause* **while True**:  
 interface.game\_mode.waiting\_screen() *# Displays the waiting screen* **try**:  
 *# This would pause the loop until it receives data if it was in blocking-mode* **if** net.receive(): *# Receives data when the game has begun so it can stop the loop* net.client.setblocking(**True**) *# Reset it so it waits to receive the player number* **except**: *# An error occurs when no data is received* **pass** *# So the loop continues without pausing* **else**:  
 player\_number = net.receive() *# Receive your player number from the server* interface.running = **False  
 break** running = **True  
  
 while** running:  
 **try**:  
 state = net.receive()  
  
 **except** Exception **as** e:  
 print(e)  
 print(**"\nRan into an issue when receiving the data."**)  
 running = **False  
  
 if** state.game.finished:  
 running = **False** interface.game\_screen.finished\_screen(state.game)  
  
 **if** state.game.turn == player\_number: *# If it's your turn you perform an action* **if** state.payload == **"Choose"**:  
 **while not** interface.card\_chosen: *# Client hasn't chosen card yet* interface.game\_screen.display(player\_number, state.game) *# To display the game screen* interface.card\_chosen = **False** *# Reset it so they can choose another card next turn* **elif** state.payload == **"Confirm"**:  
 interface.game\_screen.confirm = **True  
 while** interface.game\_screen.confirm:  
 interface.game\_screen.ask(state.game) *# Asks the user if they want to place the drawn card down* **if** state.payload == **"Choose" or** state.payload == **"Confirm"**:  
 *# == 'None' when they don't place down the card they picked up, 'Executed"' if the card was placed down* action = interface.game\_screen.action  
 net.send(action) *# Send the action to the server so it can be executed* interface.game\_screen.action = **None** *# Reset the action* alert = net.receive() *# To tell the client that the action was executed  
 # alert is needed so all clients can return to the try statement at the same time and receive the game* **else**: *# If it's not your turn* net.client.setblocking(**False**) *# Stops the client socket from breaking the loop if it receives no data* **if** state.payload == **"Choose"**: *# Displays the game screen, but they won't be able to pick a card* **while True**:  
 interface.game\_screen.display(player\_number, state.game)  
  
 **try**:  
 msg = net.receive()  
 *# When a player has made a move you receive a message which breaks the loop* **if** msg == **"Executed" or "None"**: *# If a player is confirming their action you will wait* **break** *# To return to the main loop* **except**:  
 **pass** *# Stops the loop from breaking when no data is received* **else**: *# If an action was just executed it moves on and waits to receive the game again* **pass** net.client.setblocking(**True**)  
 *# Resetting it to True, so the program waits at the top try statement until the client receives the game***if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 main()

## game.py file

**from** deck **import** Deck  
  
  
**class** Player:  
 **def** \_\_init\_\_(self, id):  
 self.deck = []  
 self.id = id  
  
  
**class** Uno:  
 **def** \_\_init\_\_(self):  
 self.direction = **"clockwise"** self.turn = 0  
  
 self.game\_mode = **None** self.winner = **None** self.finished = **False** self.pressed\_uno = **False** *# This is assigned the ID of the player that pressed the Uno button later* self.forgot\_player = **False** *# This is assigned the ID of the player that forgot to press Uno* self.player\_list = []  
 self.discard\_pile = []  
  
 self.dk = Deck()  
  
 **def** add\_player(self, player\_id):  
 self.player\_list.append(Player(player\_id)) *# Create new player* **def** change\_direction(self):  
 **if** self.direction == **"clockwise"**:  
 self.direction = **"anticlockwise"  
 else**:  
 self.direction = **"clockwise"  
  
 def** next\_turn(self):  
 *""" Increments or decrements the turn variable to determine whose turn is next """  
 # A separate variable so I can use next\_turn as an index to find the next player  
 # Or else self.turn would be incremented every time I call this method* next\_turn = self.turn  
  
 **if** self.direction == **"clockwise"**:  
 next\_turn = (self.turn + 1) % len(self.player\_list) *# Becomes 0 if it reaches 3, to stop index errors* **else**:  
 next\_turn -= 1  
 **if** next\_turn == -1: *# Prevents index errors* next\_turn = len(self.player\_list) - 1 *# E.g. so it goes back to P2 after P0 has their turn* **return** next\_turn  
  
 **def** draw\_card(self, player\_deck):  
 *""" When player chooses to draw a card """* player\_deck.append(self.dk.deck[0])  
 self.dk.deck.pop(0)  
  
  
 **def** compare\_card(self):  
 *# self.player\_list[self.next\_turn()] is the next player* current\_turn = self.turn *# Used to check if the player has placed the final card down (as self.turn changes)* **if** self.discard\_pile[-1].value == **"draw 2"**:  
 *# Next player draws 2 and their turn is skipped* **for** i **in** range(2): *# next\_turn is called so you get the index of the next player* self.draw\_card(self.player\_list[self.next\_turn()].deck)  
  
 **for** i **in** range(2): *# Turn increments twice so it skips the next player* self.turn = self.next\_turn()  
  
 **elif** self.discard\_pile[-1].value == **"skip"**:  
 **for** i **in** range(2):  
 self.turn = self.next\_turn()  
  
 **elif** self.discard\_pile[-1].value == **"reverse"**:  
 self.change\_direction()  
 self.turn = self.next\_turn()  
  
 **if** self.game\_mode == 2: *# Reverse acts as a skip card in 2 player mode* self.turn = self.next\_turn() *# Returns back to you* **elif** self.discard\_pile[-1].value == **"wild 4"**:  
 **for** i **in** range(4): *# Next player draws 4 cards* self.draw\_card(self.player\_list[self.next\_turn()].deck)  
   
 **for** i **in** range(2):  
 self.turn = self.next\_turn()  
  
 **else**: *# Normal numbered card placed down* self.turn = self.next\_turn()  
  
 **if** len(self.player\_list[current\_turn].deck) == 0: *# If a player has placed all their cards down* self.finished = **True** self.winner = current\_turn  
 **elif** len(self.player\_list[current\_turn].deck) == 1: *# If they have 1 card remaining* self.check\_for\_uno()  
  
 **def** check\_for\_uno(self):  
 *""" Checks if the player remembered to say Uno before putting down his 2nd to last card """* **if** self.forgot\_player **is not False**: *# They forgot* **for** i **in** range(4): *# They must draw 4 cards* self.draw\_card(self.player\_list[self.forgot\_player].deck)  
  
  
 **def** set\_up(self):  
 *""" Give each player 7 cards and add the card at the top of the deck to the discard pile """* self.dk.create\_deck()  
 self.dk.shuffle()  
  
 **for** player **in** self.player\_list:  
 player.deck = self.dk.deal\_cards(player.deck) *# Give the players 7 cards* **for** i **in** range(0, len(self.dk.deck)): *# So the game doesn't start with a wild card* **if** self.dk.deck[i].colour **is not None**:  
 self.discard\_pile.append(self.dk.deck[i]) *# Card at the top of the deck is placed down first (index 0)* **break  
  
 def** start\_game(self, game\_mode):  
 self.game\_mode = game\_mode  
 self.set\_up()

## actions file

**class** Response:  
 **def** \_\_init\_\_(self, game, payload):  
 self.game = game *# the new game state* self.payload = payload *# payload - might be None - this is for additional data***class** PlaceCard: *# Used to tell the server which card you want to place down* **def** \_\_init\_\_(self, choice, \*\*kw):  
 self.choice = choice *# A number representing the player's choice* self.colour = kw.get(**"colour"**)  
 self.pressed\_uno = **False  
  
 def** execute(self, game): *# The server calls this method once it receives the object  
 # Checks if the player picked a wild card (they have no colour)* **if** game.player\_list[game.turn].deck[self.choice].colour **is None**:  
 *# Replaces the wild card's 'None' colour with the colour chosen* game.player\_list[game.turn].deck[self.choice].colour = self.colour  
  
 *# Places the card down* game.discard\_pile.append(game.player\_list[game.turn].deck[self.choice]) *# Add the card to the discard pile* game.player\_list[game.turn].deck.pop(self.choice) *# Remove it from your deck* **if** len(game.player\_list[game.turn].deck) == 1 **and** self.pressed\_uno **is True**:  
 *# 1 card remaining and they pressed the uno button already* game.pressed\_uno = game.turn *# So we know which player pressed uno* **elif** len(game.player\_list[game.turn].deck) == 1 **and** self.pressed\_uno **is False**: *# They forgot to say uno* game.forgot\_player = game.turn  
  
 **elif** len(game.player\_list[game.turn].deck) > 1 **and** self.pressed\_uno **is True**:  
 game.pressed\_uno = **False** *# If the previous player pressed the Uno button  
 # Reset this variable so that it doesn't display to users that a player said 'Uno', in the next turn* **elif** game.forgot\_player **is not False**: *# If the previous player forgot to press the Uno button* game.forgot\_player = **False** *# Reset this variable so that it doesn't display to users that a player has forgotten, in the next turn* **return** Response(game, **"Executed"**)  
  
  
**class** DrawCard: *# This is sent to the server to tell them that you want to draw a card* **def** \_\_init\_\_(self):  
 **pass  
  
 def** execute(self, game):  
 game.draw\_card(game.player\_list[game.turn].deck) *# Draw 1 card* **if** game.pressed\_uno **is not False**: *# If the previous player pressed the Uno button* game.pressed\_uno = **False** *# Reset* **elif** game.forgot\_player **is not False**:  
 game.forgot\_player = **False** *# Reset  
  
 # If the card you drew is valid the player will be asked if they want to place it down immediately -* **if** game.player\_list[game.turn].deck[-1].colour == game.discard\_pile[-1].colour **or** \  
 game.player\_list[game.turn].deck[-1].value == game.discard\_pile[-1].value:  
 **return** Response(game, **"Confirm"**)  
  
 **else**: *# If the card they drew cannot be placed down, it increments the turn so the game moves on* game.turn = game.next\_turn()  
 **return** Response(game, **None**)  
  
  
**class** Decision: *# This is sent to the server to tell them whether to place the picked up card down, or not* **def** \_\_init\_\_(self, decision):  
 self.decision = decision  
  
 **def** execute(self, game):  
 **if** self.decision == **"Yes"**: *# If they chose to place down the card that they picked up* game.discard\_pile.append(game.player\_list[game.turn].deck[-1])  
 game.player\_list[game.turn].deck.pop(-1)  
 **return** Response(game, **"Executed"**)  
  
 **else**: *# Moves onto the next player without placing the card down* game.turn = game.next\_turn()  
 **return** Response(game, **None**)

## deck.py file

**import** random  
  
**class** Card:  
 **def** \_\_init\_\_(self, colour, value, \*\*kwargs):  
 self.colour = colour  
 self.value = value  
 self.image = kwargs.get(**"image"**)  
  
**class** Deck:  
 **def** \_\_init\_\_(self):  
 self.colours = [**"red"**, **"blue"**, **"green"**, **"yellow"**]  
 self.deck = []  
 self.images = Images()  
  
 **def** create\_deck(self):  
 colour\_index = 0 *# Used to get the image from the correct list inside the 2D card\_list* **for** i **in** range(0, 2): *# Each number has 2 cards for each colour (e.g Two blue-8s in a deck)* **for** num **in** range(0,13): *# Creates the cards for every colour* **for** colour **in** self.colours:  
 *# 2 special cards of each type for every colour e.g 2 red skips, 2 red reverse, 2 red draw\_2* **if** num == 10:  
 self.deck.append(Card(colour, **"draw 2"**, image=self.images.card\_list[colour\_index][num]))  
 **elif** num == 11:  
 self.deck.append(Card(colour, **"reverse"**, image=self.images.card\_list[colour\_index][num]))  
 **elif** num == 12:  
 self.deck.append(Card(colour, **"skip"**, image=self.images.card\_list[colour\_index][num]))  
 **else**: *# Creating the card objects and adding them to the deck* self.deck.append(Card(colour, num, image=self.images.card\_list[colour\_index][num]))  
 colour\_index += 1 *# Incremented so it moves onto the next colour in card\_list* colour\_index = 0 *# Reset to 0 to select from the red\_cards list again* **for** i **in** range(4): *# 4 wild cards and 4 wild 4 cards in a deck* self.deck.append(Card(**None**, **"wild 4"**, image=self.images.wild\_4))  
 self.deck.append(Card(**None**, **"wild"**, image=self.images.wild)) *# Changes colour only* **def** shuffle(self):  
 **for** i **in** range(5):  
 random.shuffle(self.deck)  
  
 **def** deal\_cards(self, player\_deck):  
 *''' Deal 7 cards to the player and remove the 7 cards from the main deck'''* **for** i **in** range(7):  
 player\_deck.append(self.deck[0])  
 self.deck.pop(0) *# Remove from the back of the deck* **return** player\_deck  
  
  
**class** Images:  
 **def** \_\_init\_\_(self):  
 self.red\_0 = (**"Uno Cards\Red 0.png"**)  
 self.red\_1 = (**"Uno Cards\Red 1.png"**)  
 self.red\_2 = (**"Uno Cards\Red 2.png"**)  
 self.red\_3 = (**"Uno Cards\Red 3.png"**)  
 self.red\_4 = (**"Uno Cards\Red 4.png"**)  
 self.red\_5 = (**"Uno Cards\Red 5.png"**)  
 self.red\_6 = (**"Uno Cards\Red 6.png"**)  
 self.red\_7 = (**"Uno Cards\Red 7.png"**)  
 self.red\_8 = (**"Uno Cards\Red 8.png"**)  
 self.red\_9 = (**"Uno Cards\Red 9.png"**)  
  
 self.red\_draw\_2 = (**"Uno Cards\Red Draw 2.png"**)  
 self.red\_reverse = (**"Uno Cards\Red Reverse.png"**)  
 self.red\_skip = (**"Uno Cards\Red Skip.png"**)  
  
 self.blue\_0 = (**"Uno Cards\Blue 0.png"**)  
 self.blue\_1 = (**"Uno Cards\Blue 1.png"**)  
 self.blue\_2 = (**"Uno Cards\Blue 2.png"**)  
 self.blue\_3 = (**"Uno Cards\Blue 3.png"**)  
 self.blue\_4 = (**"Uno Cards\Blue 4.png"**)  
 self.blue\_5 = (**"Uno Cards\Blue 5.png"**)  
 self.blue\_6 = (**"Uno Cards\Blue 6.png"**)  
 self.blue\_7 = (**"Uno Cards\Blue 7.png"**)  
 self.blue\_8 = (**"Uno Cards\Blue 8.png"**)  
 self.blue\_9 = (**"Uno Cards\Blue 9.png"**)  
  
 self.blue\_draw\_2 = (**"Uno Cards\Blue Draw 2.png"**)  
 self.blue\_reverse = (**"Uno Cards\Blue Reverse.png"**)  
 self.blue\_skip = (**"Uno Cards\Blue Skip.png"**)  
  
 self.green\_0 = (**"Uno Cards\Green 0.png"**)  
 self.green\_1 = (**"Uno Cards\Green 1.png"**)  
 self.green\_2 = (**"Uno Cards\Green 2.png"**)  
 self.green\_3 = (**"Uno Cards\Green 3.png"**)  
 self.green\_4 = (**"Uno Cards\Green 4.png"**)  
 self.green\_5 = (**"Uno Cards\Green 5.png"**)  
 self.green\_6 = (**"Uno Cards\Green 6.png"**)  
 self.green\_7 = (**"Uno Cards\Green 7.png"**)  
 self.green\_8 = (**"Uno Cards\Green 8.png"**)  
 self.green\_9 = (**"Uno Cards\Green 9.png"**)  
  
 self.green\_draw\_2 = (**"Uno Cards\Green Draw 2.png"**)  
 self.green\_reverse = (**"Uno Cards\Green Reverse.png"**)  
 self.green\_skip = (**"Uno Cards\Green Skip.png"**)  
  
 self.yellow\_0 = (**"Uno Cards\Yellow 0.png"**)  
 self.yellow\_1 = (**"Uno Cards\Yellow 1.png"**)  
 self.yellow\_2 = (**"Uno Cards\Yellow 2.png"**)  
 self.yellow\_3 = (**"Uno Cards\Yellow 3.png"**)  
 self.yellow\_4 = (**"Uno Cards\Yellow 4.png"**)  
 self.yellow\_5 = (**"Uno Cards\Yellow 5.png"**)  
 self.yellow\_6 = (**"Uno Cards\Yellow 6.png"**)  
 self.yellow\_7 = (**"Uno Cards\Yellow 7.png"**)  
 self.yellow\_8 = (**"Uno Cards\Yellow 8.png"**)  
 self.yellow\_9 = (**"Uno Cards\Yellow 9.png"**)  
  
 self.yellow\_draw\_2 = (**"Uno Cards\Yellow Draw 2.png"**)  
 self.yellow\_reverse = (**"Uno Cards\Yellow Reverse.png"**)  
 self.yellow\_skip = (**"Uno Cards\Yellow Skip.png"**)  
  
 self.wild = (**"Uno Cards\Wild.png"**)  
 self.wild\_4 = (**"Uno Cards\Wild 4.png"**)  
 self.card\_back = (**"Uno Cards\Back Of Card.png"**)  
  
 self.red\_diamond = (**"Uno Cards\Red Diamond.png"**)  
 self.blue\_diamond = (**"Uno Cards\Blue Diamond.png"**)  
 self.green\_diamond = (**"Uno Cards\Green Diamond.png"**)  
 self.yellow\_diamond = (**"Uno Cards\Yellow Diamond.png"**)  
  
 self.clockwise = (**"Uno Cards\Clockwise.png"**)  
 self.anticlockwise = (**"Uno Cards\Anticlockwise.png"**)  
  
 self.card\_list = []  
 self.red\_cards = []  
 self.green\_cards = []  
 self.blue\_cards = []  
 self.yellow\_cards = []  
  
 self.create\_card\_list()  
  
 **def** create\_card\_list(self):  
 *""" Add the images into separate lists then combine them to create a 2D List """* self.red\_cards.append(self.red\_0)  
 self.red\_cards.append(self.red\_1)  
 self.red\_cards.append(self.red\_2)  
 self.red\_cards.append(self.red\_3)  
 self.red\_cards.append(self.red\_4)  
 self.red\_cards.append(self.red\_5)  
 self.red\_cards.append(self.red\_6)  
 self.red\_cards.append(self.red\_7)  
 self.red\_cards.append(self.red\_8)  
 self.red\_cards.append(self.red\_9)  
 self.red\_cards.append(self.red\_draw\_2)  
 self.red\_cards.append(self.red\_reverse)  
 self.red\_cards.append(self.red\_skip)  
  
 self.blue\_cards.append(self.blue\_0)  
 self.blue\_cards.append(self.blue\_1)  
 self.blue\_cards.append(self.blue\_2)  
 self.blue\_cards.append(self.blue\_3)  
 self.blue\_cards.append(self.blue\_4)  
 self.blue\_cards.append(self.blue\_5)  
 self.blue\_cards.append(self.blue\_6)  
 self.blue\_cards.append(self.blue\_7)  
 self.blue\_cards.append(self.blue\_8)  
 self.blue\_cards.append(self.blue\_9)  
 self.blue\_cards.append(self.blue\_draw\_2)  
 self.blue\_cards.append(self.blue\_reverse)  
 self.blue\_cards.append(self.blue\_skip)  
  
 self.green\_cards.append(self.green\_0)  
 self.green\_cards.append(self.green\_1)  
 self.green\_cards.append(self.green\_2)  
 self.green\_cards.append(self.green\_3)  
 self.green\_cards.append(self.green\_4)  
 self.green\_cards.append(self.green\_5)  
 self.green\_cards.append(self.green\_6)  
 self.green\_cards.append(self.green\_7)  
 self.green\_cards.append(self.green\_8)  
 self.green\_cards.append(self.green\_9)  
 self.green\_cards.append(self.green\_draw\_2)  
 self.green\_cards.append(self.green\_reverse)  
 self.green\_cards.append(self.green\_skip)  
  
 self.yellow\_cards.append(self.yellow\_0)  
 self.yellow\_cards.append(self.yellow\_1)  
 self.yellow\_cards.append(self.yellow\_2)  
 self.yellow\_cards.append(self.yellow\_3)  
 self.yellow\_cards.append(self.yellow\_4)  
 self.yellow\_cards.append(self.yellow\_5)  
 self.yellow\_cards.append(self.yellow\_6)  
 self.yellow\_cards.append(self.yellow\_7)  
 self.yellow\_cards.append(self.yellow\_8)  
 self.yellow\_cards.append(self.yellow\_9)  
 self.yellow\_cards.append(self.yellow\_draw\_2)  
 self.yellow\_cards.append(self.yellow\_reverse)  
 self.yellow\_cards.append(self.yellow\_skip)  
  
 self.card\_list.append(self.red\_cards) *# Create 2D list* self.card\_list.append(self.blue\_cards)  
 self.card\_list.append(self.green\_cards)  
 self.card\_list.append(self.yellow\_cards)  
  
 self.card\_list.append(self.wild)  
 self.card\_list.append(self.wild\_4)

## buttons.py

**import** pygame  
  
  
**class** Button():  
 **def** \_\_init\_\_(self, x, y, width, height):  
 self.rect = pygame.Rect(x, y, width, height)  
 self.font = pygame.font.Font(**None**, 30)  
 self.colour\_active = pygame.Color(**"yellow"**) *#gray15* self.colour\_passive = pygame.Color(**"lightskyblue3"**)  
 self.colour = self.colour\_passive  
 self.active = **False** self.text = **""  
  
 def** change\_colour(self, pos):  
 *""" Changes colour of button if you clicked on it """* **if** self.rect.collidepoint(pos): *# Check the pos of mouse click and see if it's inside the rect* self.active = **True** *# So that we can start typing inside user\_rect* self.colour = self.colour\_active  
 **else**:  
 self.active = **False** *# Click outside of box* self.colour = self.colour\_passive  
  
 **def** get\_text(self, key):  
 *""" Gets your inputted text and stores it into the text variable """* **if** key == pygame.K\_BACKSPACE:  
 self.text = self.text[:-1] *# Till 2nd last character* **else**: *# To add the letter you pressed onto screen later* self.text += key.unicode *# Unicode is the info of the button pressed.* **def** draw\_text\_box(self, surface): *# surface = screen  
 """ Draw the rectangle onto screen """* self.input\_text = self.font.render(self.text, **True**, (255, 255, 255)) *# Put inputted text onto screen* surface.blit(self.input\_text, (self.rect.x + 5, self.rect.y + 5))  
  
 self.rect.w = max(350, self.input\_text.get\_width() + 10) *# Set rect width  
 # Max uses the largest argument, so width is 350 initially. Becomes bigger after text width > 350.* pygame.draw.rect(surface, self.colour, self.rect, 2) *# Include a border width of 2 to blit the border only* **def** draw\_rect(self, surface):  
 pygame.draw.rect(surface, self.colour, self.rect, 2)

## interface.py file

**import** pygame  
**from** login **import** \*  
**from** menu **import** \*  
**from** game\_screen **import** GameScreen  
  
  
**class** Interface:  
 **def** \_\_init\_\_(self):  
 pygame.init()  
 self.SCREEN\_W = 1200  
 self.SCREEN\_H = 1000  
 self.screen = pygame.display.set\_mode((self.SCREEN\_W, self.SCREEN\_H))  
  
 pygame.display.set\_caption(**"Uno"**)  
 self.clock = pygame.time.Clock()  
 self.font = pygame.font.Font(**None**, 30)  
 self.running = **True** self.mouse\_pos = **None** self.volume = 0.5  
  
 self.sign\_in = SignIn(self) *# Self is a parameter so the menu can change the variables in this interface* self.register = Register(self)  
  
 self.main\_menu = MainMenu(self)  
 self.options = Options(self)  
 self.game\_screen = GameScreen(self)  
 self.current\_screen = self.main\_menu *# Starts on the main\_menu screen  
 # Changes depending on what the user navigates to* self.game\_mode = GameMode(self)  
 self.game\_mode\_choice = 0  
  
 self.clicked = **False** self.pressed = **False** self.key = **None** self.card\_chosen = **False** self.ENTER\_KEY = **False** self.BACK\_KEY = **False** self.DOWN\_KEY = **False** self.UP\_KEY = **False** self.LEFT\_KEY = **False** self.RIGHT\_KEY = **False  
  
 def** check\_events(self):  
 **for** event **in** pygame.event.get():  
 **if** event.type == pygame.QUIT:  
 pygame.quit()  
  
 self.mouse\_pos = pygame.mouse.get\_pos()  
  
 **if** event.type == pygame.MOUSEBUTTONDOWN:  
 self.clicked = **True  
  
 elif** event.type == pygame.KEYDOWN: *# Press a key* self.pressed = **True** self.key = event  
  
 **if** event.key == pygame.K\_RETURN: *# If they click these keys the variable becomes True* self.ENTER\_KEY = **True** *# This is used to navigate through the Menu* **elif** event.key == pygame.K\_BACKSPACE:  
 self.BACK\_KEY = **True  
 elif** event.key == pygame.K\_DOWN:  
 self.DOWN\_KEY = **True  
 elif** event.key == pygame.K\_UP:  
 self.UP\_KEY = **True  
 elif** event.key == pygame.K\_RIGHT:  
 self.RIGHT\_KEY = **True  
 elif** event.key == pygame.K\_LEFT:  
 self.LEFT\_KEY = **True**

## login.py

This was supposed to be my login screen but as I did not manage to implement a database, this file is not being used.

**import** pygame  
**from** login **import** \*  
**from** menu **import** \*  
**from** game\_screen **import** GameScreen  
  
  
**class** Interface:  
 **def** \_\_init\_\_(self):  
 pygame.init()  
 self.SCREEN\_W = 1200  
 self.SCREEN\_H = 1000  
 self.screen = pygame.display.set\_mode((self.SCREEN\_W, self.SCREEN\_H))  
  
 pygame.display.set\_caption(**"Uno"**)  
 self.clock = pygame.time.Clock()  
 self.font = pygame.font.Font(**None**, 30)  
 self.running = **True** self.mouse\_pos = **None** self.volume = 0.5  
  
 self.sign\_in = SignIn(self) *# Self is a parameter so the menu can change the variables in this interface* self.register = Register(self)  
  
 self.main\_menu = MainMenu(self)  
 self.options = Options(self)  
 self.game\_screen = GameScreen(self)  
 self.current\_screen = self.main\_menu *# Starts on the main\_menu screen  
 # Changes depending on what the user navigates to* self.game\_mode = GameMode(self)  
 self.game\_mode\_choice = 0  
  
 self.clicked = **False** self.pressed = **False** self.key = **None** self.card\_chosen = **False** self.ENTER\_KEY = **False** self.BACK\_KEY = **False** self.DOWN\_KEY = **False** self.UP\_KEY = **False** self.LEFT\_KEY = **False** self.RIGHT\_KEY = **False  
  
 def** check\_events(self):  
 **for** event **in** pygame.event.get():  
 **if** event.type == pygame.QUIT:  
 pygame.quit()  
  
 self.mouse\_pos = pygame.mouse.get\_pos()  
  
 **if** event.type == pygame.MOUSEBUTTONDOWN:  
 self.clicked = **True  
  
 elif** event.type == pygame.KEYDOWN: *# Press a key* self.pressed = **True** self.key = event  
  
 **if** event.key == pygame.K\_RETURN: *# If they click these keys the variable becomes True* self.ENTER\_KEY = **True** *# This is used to navigate through the Menu* **elif** event.key == pygame.K\_BACKSPACE:  
 self.BACK\_KEY = **True  
 elif** event.key == pygame.K\_DOWN:  
 self.DOWN\_KEY = **True  
 elif** event.key == pygame.K\_UP:  
 self.UP\_KEY = **True  
 elif** event.key == pygame.K\_RIGHT:  
 self.RIGHT\_KEY = **True  
 elif** event.key == pygame.K\_LEFT:  
 self.LEFT\_KEY = **True**

## menu.py file

**import** pygame  
**from** button **import** Button  
**import** webbrowser  
  
  
**class** Menu:  
 **def** \_\_init\_\_(self, interface):  
 self.interface = interface  
 self.run\_display = **True** self.MID\_W, self.MID\_H = (self.interface.SCREEN\_W / 2), (self.interface.SCREEN\_H / 2)  
  
 self.text\_font = pygame.font.Font(**None**, 40)  
 self.button\_font = pygame.font.Font(**None**, 60)  
  
 self.enter\_desc = self.text\_font.render(**"Enter To Select"**, **True**, (255, 255, 255))  
 self.back\_desc = self.text\_font.render(**"Backspace To Return"**, **True**, (255, 255, 255))  
 self.arrow\_desc = self.text\_font.render(**"Arrows To Navigate"**, **True**, (255, 255, 255))  
  
 self.menu\_sound = pygame.mixer.Sound(**"Sounds\menu sound.wav"**)  
 self.menu\_sound.set\_volume(self.interface.volume)  
  
 **def** reset\_keys(self): *# Every menu can use this method* self.interface.ENTER\_KEY = **False** self.interface.BACK\_KEY = **False** self.interface.DOWN\_KEY = **False** self.interface.UP\_KEY = **False** self.interface.RIGHT\_KEY = **False** self.interface.LEFT\_KEY = **False  
  
 def** blit\_description(self):  
 self.interface.screen.blit(self.enter\_desc, (self.MID\_W + 300, self.MID\_H + 400))  
 self.interface.screen.blit(self.back\_desc, (self.MID\_W - 520, self.MID\_H + 400))  
 self.interface.screen.blit(self.arrow\_desc, (self.MID\_W - self.arrow\_desc.get\_width() / 2, self.MID\_H + 400))  
  
  
**class** MainMenu(Menu):  
 **def** \_\_init\_\_(self, interface): *# Having interface as an argument allows us to its variables and the screen* super().\_\_init\_\_(interface) *# Inherit from Menu class to access its attributes and methods* self.menu\_font = pygame.font.Font(**None**, 100)  
 self.title\_font = pygame.font.Font(**None**, 170)  
 self.play\_label = self.menu\_font.render(**"Play"**, **True**, (255, 255, 255))  
 self.option\_label = self.menu\_font.render(**"Options"**, **True**, (255, 255, 255))  
 self.rules\_label = self.menu\_font.render(**"Rules"**, **True**, (255, 255, 255))  
  
 self.title = self.title\_font.render(**"Uno"**, **True**, (255, 255, 255))  
  
 *# x, y for top left corner of the rectangle and then width, height* self.play\_button = Button(self.MID\_W - 160, self.MID\_H - 130, 320, 100)  
 self.option\_button = Button(self.MID\_W - 160, self.MID\_H + 30, 320, 100)  
 self.rules\_button = Button(self.MID\_W - 160, self.MID\_H + 190, 320, 100)  
  
 self.cursor = Button(self.MID\_W - 160, self.MID\_H - 130, 320, 100)  
 self.cursor.colour = self.cursor.colour\_active  
  
 self.button\_list = [self.play\_button, self.option\_button, self.rules\_button, self.cursor]  
  
 **def** display(self):  
 *""" Displays the text and buttons and checks if they pressed a key """* self.run\_display = **True  
 while** self.run\_display:  
 self.interface.screen.fill((0,0,0))  
 self.menu\_sound.set\_volume(self.interface.volume)  
  
 self.interface.screen.blit(self.title, (self.MID\_W - self.title.get\_width() / 2, *# This centres the text* self.MID\_H - 320))  
 self.interface.screen.blit(self.play\_label, (self.MID\_W - self.play\_label.get\_width() / 2,  
 self.MID\_H - 110))  
 self.interface.screen.blit(self.option\_label, (self.MID\_W - self.option\_label.get\_width() / 2,  
 self.MID\_H + 50))  
 self.interface.screen.blit(self.rules\_label, (self.MID\_W - self.rules\_label.get\_width() / 2,  
 self.MID\_H + 210))  
  
 self.blit\_description()  
  
 self.interface.check\_events()  
 self.check\_input()  
  
 **for** b **in** self.button\_list:  
 b.draw\_rect(self.interface.screen)  
  
 pygame.display.update()  
 self.interface.clock.tick(60) *# 60 fps* self.reset\_keys()  
  
 **def** move\_cursor(self):  
 *""" Moves the cursor up and down if you pressed the corresponding key """* **if** self.interface.DOWN\_KEY: *# If they pressed the Down arrow key* **if** self.cursor.rect.y != self.rules\_button.rect.y: *# Stops the cursor from moving above/below button* self.cursor.rect.y += 160 *# Moves the cursor up and down* pygame.mixer.Sound.play(self.menu\_sound)  
  
 **elif** self.interface.UP\_KEY:  
 **if** self.cursor.rect.y != self.play\_button.rect.y:  
 self.cursor.rect.y -= 160  
 pygame.mixer.Sound.play(self.menu\_sound)  
  
 **def** check\_input(self):  
 *""" Checks if they pressed a key """* self.move\_cursor()  
  
 **if** self.interface.ENTER\_KEY: *# If they pressed enter* pygame.mixer.Sound.play(self.menu\_sound)  
  
 **for** button **in** self.button\_list: *# Finds the button selected* **if** button.rect.y == self.cursor.rect.y: *# If the cursor rect overlaps with that button* **if** button == self.play\_button:  
 self.interface.current\_screen = self.interface.game\_mode *# Changes to game mode menu* self.run\_display = **False  
  
 elif** button == self.option\_button:  
 self.interface.current\_screen = self.interface.options  
 self.run\_display = **False  
  
 elif** button == self.rules\_button:  
 webbrowser.open(**r"https://www.ultraboardgames.com/uno/game-rules.php"**)  
  
  
  
**class** Options(Menu):  
 **def** \_\_init\_\_(self, interface):  
 super().\_\_init\_\_(interface)  
  
 self.volume\_label = self.button\_font.render(**"Volume"**, **True**, (255, 255, 255))  
 self.volume\_button = Button(self.MID\_W - 400, self.MID\_H - 135, 240, 70)  
  
 self.volume\_slider = Button(self.MID\_W - 130, self.MID\_H - 100, 500, 7)  
 self.volume\_slider.active = **True** self.volume\_slider.colour = self.volume\_slider.colour\_active  
  
 self.volume = 0.5  
 self.volume\_circle\_x = 120  
  
 self.sound\_label = self.button\_font.render(**"Sound"**, **True**, (255, 255, 255))  
 self.sound\_button = Button(self.MID\_W - 400, self.MID\_H, 240, 70)  
  
 self.sound = Button(self.volume\_slider.rect.centerx - 50, self.MID\_H, 100, 70)  
  
 **def** display(self):  
 self.run\_display = **True  
 while** self.run\_display:  
 self.interface.screen.fill((0, 0, 0))  
  
 self.menu\_sound.set\_volume(self.interface.volume)  
 **if** self.sound.active:  
 self.interface.volume = 0  
  
 self.interface.check\_events()  
 self.check\_input()  
  
 self.interface.screen.blit(self.volume\_label,  
 (self.volume\_button.rect.centerx - self.volume\_label.get\_width() / 2,  
 self.volume\_button.rect.centery - self.volume\_label.get\_height() / 2))  
  
 self.interface.screen.blit(self.sound\_label,  
 (self.sound\_button.rect.centerx - self.sound\_label.get\_width() / 2,  
 self.sound\_button.rect.centery - self.sound\_label.get\_height() / 2))  
  
 self.volume\_button.draw\_rect(self.interface.screen)  
 self.sound\_button.draw\_rect(self.interface.screen)  
  
 self.volume\_slider.draw\_rect(self.interface.screen)  
 self.sound.draw\_rect(self.interface.screen)  
  
 pygame.draw.circle(self.interface.screen, (255, 255, 255), (self.MID\_W + self.volume\_circle\_x,  
 self.volume\_slider.rect.centery), 20)  
  
 **if** self.sound.active:  
 pygame.draw.line(self.interface.screen, (255, 255, 255),  
 (self.sound.rect.topleft), (self.sound.rect.bottomright), 3)  
 pygame.draw.line(self.interface.screen, (255, 255, 255),  
 (self.sound.rect.topright), (self.sound.rect.bottomleft), 3)  
  
 self.blit\_description()  
  
 self.interface.clock.tick(60) *# 60 fps* pygame.display.update()  
 self.reset\_keys()  
  
 **def** check\_input(self): *# Create sliders or options later* **if** self.interface.BACK\_KEY:  
 pygame.mixer.Sound.play(self.menu\_sound)  
 self.interface.current\_screen = self.interface.main\_menu  
 self.run\_display = **False  
  
 elif** self.interface.LEFT\_KEY **and** self.volume\_slider.active **and** (self.MID\_W + self.volume\_circle\_x !=  
 self.MID\_W - 130):  
 pygame.mixer.Sound.play(self.menu\_sound)  
 self.volume\_circle\_x -= 50  
 self.volume -= 0.1 *# Decrease the volume* **if not** self.sound.active: *# Only changes the volume if the player has enabled the sound* self.interface.volume = self.volume  
  
 **elif** self.interface.RIGHT\_KEY **and** self.volume\_slider.active **and** (self.MID\_W + self.volume\_circle\_x !=  
 self.MID\_W + 370):  
 pygame.mixer.Sound.play(self.menu\_sound)  
 self.volume\_circle\_x += 50  
 self.volume += 0.1 *# Increase the volume* **if not** self.sound.active:  
 self.interface.volume = self.volume  
  
 **elif** self.interface.UP\_KEY **and** self.sound.colour == self.sound.colour\_active:  
 pygame.mixer.Sound.play(self.menu\_sound)  
 self.volume\_slider.active = **True** self.volume\_slider.colour = self.volume\_slider.colour\_active  
  
 self.sound.colour = self.sound.colour\_passive  
  
 **elif** self.interface.DOWN\_KEY **and** self.volume\_slider.active:  
 pygame.mixer.Sound.play(self.menu\_sound)  
 self.sound.colour = self.sound.colour\_active  
  
 self.volume\_slider.active = **False** self.volume\_slider.colour = self.volume\_slider.colour\_passive  
  
 **elif** self.interface.ENTER\_KEY **and** self.sound.colour == self.sound.colour\_active:  
 **if** self.sound.active: *# To enable or disable the sound* self.sound.active = **False** self.interface.volume = self.volume  
 **else**:  
 self.sound.active = **True** self.interface.volume = 0  
  
**class** GameMode(Menu):  
 **def** \_\_init\_\_(self, interface):  
 super().\_\_init\_\_(interface)  
 self.waiting = **False** self.waiting\_text = self.button\_font.render(**"Waiting For Other Players"**, **True**, (255, 255, 255))  
  
 self.two\_player\_label = self.button\_font.render(**"Two Player"**, **True**, (255, 255, 255))  
 self.two\_player\_button = Button(self.MID\_W - 150, self.MID\_H - 120, 300, 80) *# x, y, width, height* self.three\_player\_label = self.button\_font.render(**"Three Player"**, **True**, (255, 255, 255))  
 self.three\_player\_button = Button(self.MID\_W - 150, self.MID\_H, 300, 80)  
  
 self.four\_player\_label = self.button\_font.render(**"Four Player"**, **True**, (255, 255, 255))  
 self.four\_player\_button = Button(self.MID\_W - 150, self.MID\_H + 120, 300, 80)  
  
 self.cursor = Button(self.MID\_W - 150, self.MID\_H, 300, 80)  
 self.cursor.colour = self.cursor.colour\_active  
  
 self.button\_list = [self.two\_player\_button, self.three\_player\_button, self.four\_player\_button, self.cursor]  
  
 self.start\_sound = pygame.mixer.Sound(**"Sounds\start sound.ogg"**)  
  
 **def** display(self):  
 self.run\_display = **True  
 while** self.run\_display:  
 self.interface.screen.fill((0,0,0))  
 self.menu\_sound.set\_volume(self.interface.volume)  
 self.start\_sound.set\_volume(self.interface.volume)  
  
 self.interface.screen.blit(self.two\_player\_label, (self.MID\_W - self.two\_player\_label.get\_width() / 2,  
 self.MID\_H - 100))  
 self.interface.screen.blit(self.three\_player\_label, (self.MID\_W - self.three\_player\_label.get\_width() / 2,  
 self.MID\_H + 20))  
 self.interface.screen.blit(self.four\_player\_label, (self.MID\_W - self.four\_player\_label.get\_width() / 2,  
 self.MID\_H + 140))  
  
 self.blit\_description()  
  
 self.interface.check\_events()  
 self.check\_input()  
  
 **for** b **in** self.button\_list:  
 b.draw\_rect(self.interface.screen)  
  
 pygame.display.update()  
 self.interface.clock.tick(60) *# 60 fps* self.reset\_keys()  
  
 **def** move\_cursor(self):  
 **if** self.interface.DOWN\_KEY:  
 **if** self.cursor.rect.y != self.four\_player\_button.rect.y: *# Stops the cursor from moving away from buttons* self.cursor.rect.y += 120  
 pygame.mixer.Sound.play(self.menu\_sound)  
  
 **if** self.interface.UP\_KEY:  
 **if** self.cursor.rect.y != self.two\_player\_button.rect.y:  
 self.cursor.rect.y -= 120  
 pygame.mixer.Sound.play(self.menu\_sound)  
  
 **def** check\_input(self):  
 self.move\_cursor()  
  
 **if** self.interface.ENTER\_KEY: *# If they pressed enter* **for** button **in** self.button\_list: *# Finds the button selected* **if** button.rect.y == self.cursor.rect.y: *# If the cursor rect overlaps with that button* **if** button == self.two\_player\_button: *# Sets the chosen game mode to a variable* self.interface.game\_mode\_choice = 2 *# Their choice will be sent to the server after* **elif** button == self.three\_player\_button:  
 self.interface.game\_mode\_choice = 3  
  
 **elif** button == self.four\_player\_button:  
 self.interface.game\_mode\_choice = 4  
  
 self.run\_display = **False** pygame.mixer.Sound.play(self.start\_sound)  
  
 **elif** self.interface.BACK\_KEY: *# Goes back to the main menu* self.interface.current\_screen = self.interface.main\_menu  
 self.run\_display = **False** pygame.mixer.Sound.play(self.menu\_sound)  
  
 **def** waiting\_screen(self):  
 *""" This is displayed while you wait for other players to join """* self.interface.screen.fill((0,0,0))  
 self.interface.check\_events()  
 self.interface.screen.blit(self.waiting\_text, (self.MID\_W - self.waiting\_text.get\_width() / 2,  
 self.MID\_H - self.waiting\_text.get\_height() / 2))  
  
 pygame.display.update()  
 self.interface.clock.tick(60) *# 60 fps* self.reset\_keys()

## game\_screen.py

**import** pygame  
**from** button **import** Button  
**from** deck **import** Images  
**from** actions **import** \*  
**from** menu **import** \*  
  
**class** Image:  
 **def** \_\_init\_\_(self, image, \*\*kw):  
 self.image = image  
 self.x = **None** self.y = **None** self.colour = kw.get(**"colour"**)  
  
**class** GameScreen(Menu):  
 **def** \_\_init\_\_(self, interface):  
 super().\_\_init\_\_(interface)  
  
 self.offset = -320  
 self.opponent\_x\_offset = -480  
 self.opponent\_y\_offset = -340  
  
 self.b\_img = pygame.image.load(Images().card\_back).convert\_alpha()  
 self.back\_image = pygame.transform.scale(self.b\_img, (self.b\_img.get\_width() \* 0.22,  
 self.b\_img.get\_height() \* 0.22))  
  
 self.red\_d = pygame.image.load(Images().red\_diamond).convert\_alpha()  
 self.blue\_d = pygame.image.load(Images().blue\_diamond).convert\_alpha()  
 self.green\_d = pygame.image.load(Images().green\_diamond).convert\_alpha()  
 self.yellow\_d = pygame.image.load(Images().yellow\_diamond).convert\_alpha()  
  
 self.draw\_button = Button(self.MID\_W - 100, self.MID\_H + 185, 200, 60)  
 self.draw\_text = self.text\_font.render(**"Draw"**, **True**, (0, 0, 0)) *#* self.uno\_button = Button(self.MID\_W - 180, self.MID\_H - 30, 70, 60)  
  
 self.confirm = **False** self.ask\_cursor = Button(self.MID\_W + 100, self.MID\_H + 100, 300, 80)  
 self.ask\_cursor.colour = self.ask\_cursor.colour\_active  
  
 self.yes\_button = Button(self.MID\_W + 100, self.MID\_H + 100, 300, 80)  
 self.yes\_text = self.button\_font.render(**"Yes"**, **True**, (255, 255, 255))  
 self.no\_button = Button(self.MID\_W - 400, self.MID\_H + 100, 300, 80)  
 self.no\_text = self.button\_font.render(**"No"**, **True**, (255, 255, 255))  
  
 self.place\_card\_sound = pygame.mixer.Sound(**"Sounds\place card.wav"**)  
 self.invalid\_card\_sound = pygame.mixer.Sound(**"Sounds\invalid card.wav"**)  
 self.button\_sound = pygame.mixer.Sound(**"Sounds\pressed button.wav"**)  
 self.finished\_sound = pygame.mixer.Sound(**"Sounds\game\_over.wav"**)  
 self.sound\_list = [self.place\_card\_sound, self.invalid\_card\_sound, self.button\_sound, self.finished\_sound]  
  
 self.deck = **None** *# Assigned later* self.action = **None** self.player\_id = **None** self.game = **None** self.colour\_cursor = **None** self.new\_colour = **None** self.choosing\_colour = **False** self.invalid = **False** self.pressed\_uno = **False** self.played\_sound = **False** self.left\_opponent = **None** self.right\_opponent = **None** self.top\_opponent = **None** self.TOP\_ID = **None** self.image\_list = []  
 self.image\_list\_length = 7  
 self.number\_of\_cards\_changed = **False** self.total\_image\_width = **None** *# Used to center all your images* self.cursor\_rect = **None  
  
 def** create\_images(self):  
 *""" Creates the images for all the cards in your deck """* self.image\_list = []  
 **for** card **in** self.deck:  
 self.image\_list.append(Image(self.scale\_image(card))) *# Create Image object and add it to list* **def** scale\_image(self, card):  
 *""" Gets the image from the image attribute in the card object and scales it """* img = pygame.image.load(card.image).convert\_alpha()  
 *# Change size of image* scaled\_image = pygame.transform.scale(img, (img.get\_width() \* 0.3, img.get\_height() \* 0.3))  
 **return** scaled\_image  
  
 **def** display\_direction(self):  
 *""" Display the direction image onto screen """* clockwise = pygame.image.load(Images().clockwise).convert\_alpha()  
 clockwise\_img = pygame.transform.scale(clockwise, (clockwise.get\_width() \* 0.15, clockwise.get\_height() \* 0.15))  
  
 anticlockwise = pygame.image.load(Images().anticlockwise).convert\_alpha()  
 anticlockwise\_img = pygame.transform.scale(anticlockwise, (anticlockwise.get\_width() \* 0.40,  
 anticlockwise.get\_height() \* 0.40))  
  
 *# This puts the image in the centre of the screen* CW\_X, ACW\_X = self.MID\_W - clockwise\_img.get\_width() / 2, self.MID\_W - anticlockwise\_img.get\_width() / 2  
 CW\_Y, ACW\_Y = self.MID\_H - clockwise\_img.get\_height() / 2, self.MID\_H - anticlockwise\_img.get\_height() / 2  
  
 **if** self.TOP\_ID **is not None**: *# 2 or 4 player mode* CW\_X += 200 *# The position is changed depending on the game mode* ACW\_X += 200  
 **else**: *# The direction arrow image is in a different position for 3 player mode* CW\_Y -= 220  
 ACW\_Y -= 220  
  
 **if** self.game.direction == **"clockwise"**:  
 self.interface.screen.blit(clockwise\_img, (CW\_X, CW\_Y))  
 **else**:  
 self.interface.screen.blit(anticlockwise\_img, (ACW\_X, ACW\_Y))  
  
 **def** display\_player\_info(self):  
 *""" Display the current player and the player number of each player """* turn\_text = self.button\_font.render(**"Current Turn: P"** + str(self.game.turn), **True**, (0, 0, 0))  
  
 my\_id\_text = self.button\_font.render(**"P"** + str(self.player\_id), **True**, (0, 0, 0))  
 self.interface.screen.blit(my\_id\_text, (self.MID\_W - my\_id\_text.get\_width() / 2, self.MID\_H + 125))  
  
 **if** self.game.game\_mode == 2:  
 self.TOP\_ID = (self.player\_id + 1) % len(self.game.player\_list) *# Either 0 or 1* **else**: *# Only game mode 3 and 4 have players on the left and right* **if** self.game.game\_mode == 4: *# E.g. The top player will be P1 if you're P3, or P2 if you're P4* self.TOP\_ID = self.game.player\_list.index(self.game.player\_list[self.player\_id - 2])  
  
 *# The player id's of the opponents* LEFT\_ID = (self.player\_id + 1) % len(self.game.player\_list) *# % To get the 1st index when needed  
 # e.g In 3 player mode, if it's player 2, the left becomes player 0: (2+1) % 3 = 0  
  
 # If you are P0 the right opponent will be P2 instead of P -1 (no -ve index)* RIGHT\_ID = self.game.player\_list.index(self.game.player\_list[self.player\_id - 1])  
  
 self.left\_opponent = self.game.player\_list[LEFT\_ID]  
 self.right\_opponent = self.game.player\_list[RIGHT\_ID]  
  
 right\_text = self.button\_font.render(**"P"** + str(RIGHT\_ID), **True**, (0, 0, 0))  
 left\_text = self.button\_font.render(**"P"** + str(LEFT\_ID), **True**, (0, 0, 0))  
  
 self.interface.screen.blit(right\_text, (self.MID\_W + 410, self.MID\_H - 380))  
 self.interface.screen.blit(left\_text, (self.MID\_W - 460, self.MID\_H - 380))  
  
 **if** self.TOP\_ID **is not None**: *# If it's game mode 2 or 4 then there is a player at the top* self.top\_opponent = self.game.player\_list[self.TOP\_ID]  
 top\_text = self.button\_font.render(**"P"** + str(self.TOP\_ID), **True**, (0, 0, 0))  
 self.interface.screen.blit(top\_text, (self.MID\_W - top\_text.get\_width() / 2, self.MID\_H - 380))  
  
 self.interface.screen.blit(turn\_text, (self.MID\_W - turn\_text.get\_width() / 2, self.MID\_H - 480))  
 **else**:  
 self.interface.screen.blit(turn\_text, (self.MID\_W - turn\_text.get\_width() / 2, self.MID\_H - 380))  
  
 *# If a player put a wildcard down, display the colour that was chosen for the next player* **if** self.game.discard\_pile[-1].value == **"wild" or** self.game.discard\_pile[-1].value == **"wild 4"**:  
 colour\_text = self.text\_font.render(**"Colour Chosen: "** + self.game.discard\_pile[-1].colour,  
 **True**, (0, 0, 0))  
 **if** self.TOP\_ID **is not None**: *# 2/4 P Mode uses different coordinates* self.interface.screen.blit(colour\_text, (self.MID\_W - colour\_text.get\_width() / 2, self.MID\_H - 430))  
 **else**:  
 self.interface.screen.blit(colour\_text, (self.MID\_W - colour\_text.get\_width() / 2, self.MID\_H - 335))  
  
 **def** display\_center\_card(self):  
 *""" Display in the centre the card placed at the top of the discard pile """* top\_card = Image(self.scale\_image(self.game.discard\_pile[-1])) *# Create an image object* top\_card\_img = top\_card.image *# Get the image from that card* self.interface.screen.blit(top\_card\_img, (self.MID\_W - top\_card\_img.get\_width() / 2,  
 self.MID\_H - top\_card\_img.get\_height() / 2))  
  
 **def** reset\_offsets(self):  
 *""" To display the cards in the correct position in the next loop of the while loop """* self.offset = 0  
 self.opponent\_x\_offset = -480  
 self.opponent\_y\_offset = -320  
  
 **def** display\_opponents\_cards(self):  
 *""" Displaying the opponent's cards faced down """* **if** self.game.game\_mode != 2: *# Only game mode 3 and 4 has players on the left and right side* **for** i **in** range(0, len(self.left\_opponent.deck)): *# Blit left opponent's cards* self.interface.screen.blit(self.back\_image,  
 (self.MID\_W + self.opponent\_x\_offset, self.MID\_H + self.opponent\_y\_offset))  
 self.opponent\_y\_offset += 50 *# So the cards move downwards* self.reset\_offsets() *# Reset it for the other opponent* self.opponent\_x\_offset = 480 - self.back\_image.get\_width() *# To blit the cards of the opponent on the right* **for** j **in** range(0, len(self.right\_opponent.deck)): *# # Blit right opponent's cards* self.interface.screen.blit(self.back\_image, (self.MID\_W + self.opponent\_x\_offset,  
 self.MID\_H + self.opponent\_y\_offset))  
 self.opponent\_y\_offset += 50 *# So the cards move downwards* **if** self.game.game\_mode == 2 **or** self.game.game\_mode == 4: *# Only game mode 2 and 4 have player at the top* image\_width = self.back\_image.get\_width() \* len(self.top\_opponent.deck) - \  
 ((self.back\_image.get\_width() - 60) \* len(self.top\_opponent.deck))  
  
 top\_offset = 0  
 **for** i **in** range(0, len(self.top\_opponent.deck)):  
 self.interface.screen.blit(self.back\_image,  
 ((self.MID\_W - image\_width / 2) + top\_offset, self.MID\_H - 320))  
 top\_offset += 50  
  
 **def** display\_your\_cards(self):  
 *""" Displaying your cards on the screen """  
  
 # The width from the left side of your first card to the right side of your last card (To keep them centered)* self.total\_image\_width = (self.image\_list[0].image.get\_width() \* len(self.image\_list)) - \  
 ((self.image\_list[0].image.get\_width() - 90) \* len(self.image\_list))  
  
 **for** img **in** self.image\_list: *# Go through the Image objects in the image\_list and set their co-ordinates  
 # Blit your cards # self.offset is 0 initially then is incremented each time so the cards overlap  
 # You cannot get an image's co-ordinates so I assigned their co-ordinates to an attribute to use them later* img.x = self.MID\_W - (self.total\_image\_width / 2) + self.offset  
 img.y = self.MID\_H + 270 *# Assign values to the Image object's x and y attributes* self.interface.screen.blit(img.image, (img.x, img.y))  
 self.offset += 90  
  
 **def** display(self, player\_id, game):  
 *""" Displays all the cards onto the screen and allows you to select a card if it's your turn """* self.player\_id = player\_id  
 self.game = game  
  
 self.interface.screen.fill(pygame.Color(**"darkorange"**))  
 self.set\_volume()  
 *#self.interface.screen.fill((0, 85, 255)) #(0, 100, 255)  
  
 #chocolate1 #cyan3 #dodgerblue #lightgoldenrod* self.interface.check\_events() *# Check for key presses* self.deck = self.game.player\_list[self.player\_id].deck  
 self.create\_images() *# Getting images for every card in your deck* self.display\_player\_info()  
 self.display\_direction()  
  
 **if** len(self.image\_list) != self.image\_list\_length: *# A card was placed down or drawn* self.image\_list\_length = len(self.image\_list) *# Used to adjust the cursor rect coordinates* self.number\_of\_cards\_changed = **True** self.display\_opponents\_cards()  
 self.display\_your\_cards()  
 self.display\_center\_card()  
 self.display\_uno()  
  
 self.draw\_button.draw\_rect(self.interface.screen) *# Put draw button and text onto screen* self.interface.screen.blit(self.draw\_text, (self.MID\_W - self.draw\_text.get\_width() / 2, self.MID\_H + 205))  
  
 *# Allows them to select a card if it's their turn* **if** self.game.turn == self.player\_id:  
 **if not** self.cursor\_rect **or** self.number\_of\_cards\_changed: *# Create the cursor rect.* self.cursor\_rect = self.image\_list[0].image.get\_rect(topleft=(self.MID\_W - (self.total\_image\_width / 2),  
 self.MID\_H + 270))  
 *# Creates a rectangle around the first card so you can move this cursor left and right to select a card* self.number\_of\_cards\_changed = **False** pygame.draw.rect(self.interface.screen, pygame.Color(**"black"**), self.cursor\_rect, 2) *# Draw cursor rect* self.check\_input() *# Checks for key presses and lets them move the cursor and select a card* **if** self.invalid: *# If the card they chose cannot be placed down* invalid\_text = self.text\_font.render(**"Choose Another Card"**, **True**, (255, 255, 255))  
 self.interface.screen.blit(invalid\_text, (self.MID\_W - invalid\_text.get\_width() / 2, self.MID\_H - 130))  
  
 **else**: *# If it's not their turn, they cannot perform any actions as there is no cursor rectangle* **pass** self.interface.clock.tick(60) *# 60 fps* pygame.display.update()  
 self.reset\_keys() *# Allows the user to press another key* self.reset\_offsets()  
 self.image\_list = [] *# Reset it since the user may have drawn or placed down a card* **def** check\_input(self):  
 *""" Checks if they have pressed a key and performs the necessary actions """* self.move\_cursor()  
  
 **if** self.interface.ENTER\_KEY **and not** self.draw\_button.active **and not** self.uno\_button.active: *# They chose a card* **for** image **in** self.image\_list:  
 **if** self.cursor\_rect.x == image.x: *# If the cursor overlaps with the image's rectangle* card\_index = self.image\_list.index(image) *# The pos of the chosen card in the image list (and deck)* self.choose\_card(card\_index) *# Uses the index to check if that card in the deck is valid* **if** self.interface.card\_chosen: *# They chose a valid card* self.invalid = **False** self.deck.pop(card\_index) *# So the card doesn't get displayed among your deck again* pygame.mixer.Sound.play(self.place\_card\_sound)  
 **else**: *# The card they picked was invalid* pygame.mixer.Sound.play(self.invalid\_card\_sound)  
  
 **elif** self.interface.ENTER\_KEY **and** self.uno\_button.active: *# The user pressed the uno button* pygame.mixer.Sound.play(self.button\_sound)  
 self.pressed\_uno = **True** self.uno\_button.active = **False** *# Resetting it* self.uno\_button.colour = self.uno\_button.colour\_passive  
 self.cursor\_rect.y -= 1000  
  
 **elif** self.interface.ENTER\_KEY **and** self.draw\_button.active: *# The user chose to Draw a card* pygame.mixer.Sound.play(self.button\_sound)  
 self.action = DrawCard() *# To tell the server after that the player wants to draw a card* self.interface.card\_chosen = **True** *# Stops the loop in the client* self.draw\_button.active = **False** *# Resetting it* self.draw\_button.colour = self.draw\_button.colour\_passive  
 self.cursor\_rect.y -= 1000  
  
 self.played\_sound = **False** *# Reset it so the can sound play again next turn* **def** choose\_card(self, choice):  
 *""" Checks if your chosen card is valid and adds your selected card into an action object which will be sent  
 to the server """* current\_player = self.game.player\_list[self.player\_id]  
  
 **if** current\_player.deck[choice].colour **is None**: *# If they chose a wildcard (it has no colour)* self.choosing\_colour = **True** self.reset\_keys()  
 **while** self.choosing\_colour: *# Lets the user select the next colour* self.choose\_colour()  
  
 *# Display the chosen colour* self.action = PlaceCard(choice, colour=self.new\_colour) *# Colour is an optional parameter* self.interface.card\_chosen = **True** self.new\_colour = **None** *# Reset* **elif** (current\_player.deck[choice].colour == self.game.discard\_pile[-1].colour) **or** \  
 (current\_player.deck[choice].value == self.game.discard\_pile[-1].value): *# If they chose any other card* self.action = PlaceCard(choice)  
 self.interface.card\_chosen = **True  
  
 else**: *# The card they pick does not match in colour or value* self.invalid = **True** *# They will be asked to choose another card* **if** self.pressed\_uno:  
 self.action.pressed\_uno = **True** *# To tell the server that they pressed uno* self.pressed\_uno = **False** *# Reset the variable so another user can press the uno button* **def** move\_cursor(self):  
 *""" Move the cursor rectangle left or right to select a card, or up and down to press the other buttons """* **if** self.interface.LEFT\_KEY **and not** self.draw\_button.active **and not** self.uno\_button.active: *# If they pressed left or right* **if** self.cursor\_rect.x == self.image\_list[0].x: *# If they press left while on the 1st card* self.cursor\_rect.x = self.image\_list[-1].x *# Move cursor to the last card* **else**: *# If the cursor is not on the left-most image* self.cursor\_rect.x -= 90 *# Moves the cursor rectangle* **elif** self.interface.RIGHT\_KEY **and not** self.draw\_button.active **and not** self.uno\_button.active:  
 **if** self.cursor\_rect.x == self.image\_list[-1].x: *# If the cursor is on the left-most card* self.cursor\_rect.x = self.image\_list[0].x *# Move it onto the 1st card* **else**:  
 self.cursor\_rect.x += 90  
  
 **if** self.interface.UP\_KEY **and not** self.draw\_button.active **and not** self.uno\_button.active:  
 self.draw\_button.colour = self.draw\_button.colour\_active  
 self.draw\_button.active = **True** self.cursor\_rect.y += 1000 *# Move it out of the screen* **elif** self.interface.UP\_KEY **and** self.draw\_button.active:  
 self.uno\_button.colour = self.uno\_button.colour\_active  
 self.uno\_button.active = **True** self.draw\_button.colour = self.draw\_button.colour\_passive  
 self.draw\_button.active = **False  
  
 if** self.interface.DOWN\_KEY **and** self.draw\_button.active:  
 self.cursor\_rect.y -= 1000 *# Bring it back onto screen* self.draw\_button.active = **False** self.draw\_button.colour = self.draw\_button.colour\_passive  
  
 **elif** self.interface.DOWN\_KEY **and** self.uno\_button.active:  
 self.uno\_button.colour = self.uno\_button.colour\_passive  
 self.uno\_button.active = **False** self.draw\_button.colour = self.draw\_button.colour\_active  
 self.draw\_button.active = **True  
  
 def** set\_volume(self):  
 **for** sound **in** self.sound\_list:  
 sound.set\_volume(self.interface.volume)  
  
 **def** ask(self, game):  
 *""" If the card they drew is valid the user is asked if they want to place it down """* self.interface.screen.fill((0, 100, 255))  
 self.interface.check\_events()  
  
 ask\_text = self.button\_font.render(**"Place The Card Down?"**, **True**, (255, 255, 255))  
 self.interface.screen.blit(ask\_text, (self.MID\_W - ask\_text.get\_width() / 2, self.MID\_H - 400))  
  
 your\_player = game.player\_list[self.player\_id]  
 picked\_up\_card = self.scale\_image(your\_player.deck[-1]) *# The card you just picked up is at index -1* self.interface.screen.blit(picked\_up\_card, (self.MID\_W - picked\_up\_card.get\_width() / 2, self.MID\_H - 200))  
  
 self.interface.screen.blit(self.yes\_text, (self.MID\_W + 215, self.MID\_H + 125)) *# Drawing the text* self.interface.screen.blit(self.no\_text, (self.MID\_W - 275, self.MID\_H + 125))  
 self.yes\_button.draw\_rect(self.interface.screen) *# Putting buttons onto screen* self.no\_button.draw\_rect(self.interface.screen)  
  
 **if** self.interface.LEFT\_KEY: *# Moves the cursor rect so you can select an option* **if** self.ask\_cursor.rect.x != self.no\_button.rect.x: *# If it's not already on the 'no' button* self.ask\_cursor.rect.x -= 500  
 **elif** self.interface.RIGHT\_KEY:  
 **if** self.ask\_cursor.rect.x != self.yes\_button.rect.x:  
 self.ask\_cursor.rect.x += 500  
  
 **elif** self.interface.ENTER\_KEY:  
 **if** self.ask\_cursor.rect.x == self.yes\_button.rect.x:  
 self.action = Decision(**"Yes"**) *# This will be sent to the server to inform it of your decision* **else**:  
 self.action = Decision(**"No"**)  
  
 self.confirm = **False** *# To stop the loop in the client* self.ask\_cursor.draw\_rect(self.interface.screen)  
 pygame.display.update()  
 self.interface.clock.tick(60)  
 self.reset\_keys()  
  
 **def** choose\_colour(self):  
 self.interface.screen.fill((pygame.Color(**"cornflowerblue"**)))  
 self.interface.check\_events()  
 font = pygame.font.Font(**None**, 80)  
  
 choose\_text = font.render(**"Choose A Colour"**, **True**, (0, 0, 0))  
 self.interface.screen.blit(choose\_text, (self.MID\_W - choose\_text.get\_width() / 2, self.MID\_H - 300))  
  
 *# Create an image object for each diamond and assign their images and co-ordinates to attributes* red\_diamond = Image(pygame.transform.scale(self.red\_d, (self.red\_d.get\_width() \* 0.6,  
 self.red\_d.get\_height() \* 0.6)), colour=**"red"**)  
 diamond\_width = red\_diamond.image.get\_width()  
 red\_diamond.x, red\_diamond.y = self.MID\_W - (75 + diamond\_width \* 2), self.MID\_H - diamond\_width / 2  
  
 blue\_diamond = Image(pygame.transform.scale(self.blue\_d, (self.blue\_d.get\_width() \* 0.6,  
 self.blue\_d.get\_height() \* 0.6)), colour=**"blue"**)  
 blue\_diamond.x, blue\_diamond.y = self.MID\_W - (25 + diamond\_width), self.MID\_H - diamond\_width / 2  
  
 green\_diamond = Image(pygame.transform.scale(self.green\_d, (self.green\_d.get\_width() \* 0.6,  
 self.green\_d.get\_height() \* 0.6)), colour=**"green"**)  
 green\_diamond.x, green\_diamond.y = self.MID\_W + 25, self.MID\_H - diamond\_width / 2  
  
 yellow\_diamond = Image(pygame.transform.scale(self.yellow\_d, (self.yellow\_d.get\_width() \* 0.6,  
 self.yellow\_d.get\_height() \* 0.6)), colour=**"yellow"**)  
 yellow\_diamond.x, yellow\_diamond.y = self.MID\_W + 75 + diamond\_width, self.MID\_H - diamond\_width / 2  
  
 diamond\_list = [red\_diamond, blue\_diamond, green\_diamond, yellow\_diamond]  
  
 **for** diamond **in** diamond\_list: *# Blit all 4 diamonds* self.interface.screen.blit(diamond.image, (diamond.x, diamond.y))  
  
 **if not** self.colour\_cursor: *# If hasn't been drawn yet - stops the cursor from being drawn at initial position* self.colour\_cursor = red\_diamond.image.get\_rect(topleft=(self.MID\_W - (75 + diamond\_width \* 2),  
 self.MID\_H - diamond\_width / 2))  
  
 **if** self.interface.LEFT\_KEY:  
 **if** self.colour\_cursor.x == diamond\_list[0].x: *# If the cursor is at the first card (leftmost)* self.colour\_cursor.x = diamond\_list[-1].x *# Move cursor from the left side to the right* **else**:  
 self.colour\_cursor.x -= (50 + diamond\_width)  
  
 **elif** self.interface.RIGHT\_KEY:  
 **if** self.colour\_cursor.x == diamond\_list[-1].x:  
 self.colour\_cursor.x = diamond\_list[0].x *# Move cursor from the right side to the left* **else**:  
 self.colour\_cursor.x += (50 + diamond\_width)  
  
 **elif** self.interface.ENTER\_KEY: *# If they selected a colour* pygame.mixer.Sound.play(self.button\_sound)  
  
 **for** diamond **in** diamond\_list:  
 **if** self.colour\_cursor.x == diamond.x: *# Find the selected colour* self.new\_colour = diamond.colour  
 self.choosing\_colour = **False** pygame.draw.rect(self.interface.screen, pygame.Color(**"gray25"**), self.colour\_cursor, 3)  
  
 self.interface.clock.tick(60)  
 self.reset\_keys()  
 pygame.display.update()  
  
 **def** display\_uno(self):  
 self.uno\_button.draw\_rect(self.interface.screen)  
  
 uno\_font = pygame.font.Font(**None**, 30)  
 uno\_text = uno\_font.render(**"Uno"**, **True**, (0, 0, 0))  
 self.interface.screen.blit(uno\_text, (self.uno\_button.rect.centerx - uno\_text.get\_width() / 2,  
 self.uno\_button.rect.centery - uno\_text.get\_height() / 2))  
  
 **if** self.game.pressed\_uno **is not False**: *# A player pressed the uno button* uno\_player\_text = self.button\_font.render(**f"P{**str(self.game.pressed\_uno)**} said Uno!"**, **True**, (0, 0, 0))  
 self.interface.screen.blit(uno\_player\_text, (self.MID\_W + 250, self.MID\_H - 480))  
  
 **elif** self.game.forgot\_player **is not False**: *# A player forgot to press the uno button* forgot\_text = self.button\_font.render(**f"P{**str(self.game.forgot\_player)**} didn't say Uno!"**, **True**, (0, 0, 0))  
 self.interface.screen.blit(forgot\_text, (self.MID\_W + 250, self.MID\_H - 480))  
  
 **def** finished\_screen(self, game):  
 x = 0  
 pygame.mixer.Sound.play(self.finished\_sound)  
 **while** x != 500: *# So the window closes after the screen is displayed for a short amount of time* self.interface.screen.fill((0, 85, 255))  
 self.interface.check\_events()  
  
 finished\_text = self.button\_font.render(**"The game has finished!"**, **True**, (0, 0, 0))  
 winner\_text = self.button\_font.render(**"The winner is P"** + str(game.winner), **True**, (0, 0, 0))  
  
 self.interface.screen.blit(finished\_text, (self.MID\_W - finished\_text.get\_width() / 2, self.MID\_H - 300))  
 self.interface.screen.blit(winner\_text, (self.MID\_W - winner\_text.get\_width() / 2, self.MID\_H - 200))  
  
 self.interface.clock.tick(60)  
 pygame.display.update()  
 x += 1  
  
 pygame.quit()

# 4.0 – Testing

The youtube video link for my testing is: https://www.youtube.com/watch?v=\_gqZJwdrH\_g

|  |  |  |  |
| --- | --- | --- | --- |
| **Test** | **Expected Outcome** | **Successful (Y/N)** | **Time Stamp** |
| Navigating through the menu with the keyboard | The cursor rectangle should move and the buttons on the menu should all work | Y | 00:00 |
| Selecting a game mode | When the game begins the game mode should be the one selected | Y | 00:00 |
| Rules button | Selecting this button should open a website which explains the rules | Y | 00:00 |
| If the Options button, volume slider and sound button works | Adjusting the volume slider should change the volume. Clicking the sound button should enable or disable the sound | Y | 00:00 |
| If the deck is being shuffled | The deck should be shuffled before the game begins so that you get random cards. | Y | 0:55 |
| If each player is being dealt 7 cards | When the game begins each player should have 7 cards | Y | 0:55 |
| If you can see your opponent’s cards as well as your own | The opponent’s cards should be placed faced down in contrast to yours. | Y | 0:55 |
| If you can place a card down | Pressing enter to select a valid card should place it down | Y | 0:55 |
| Placing a normal numbered card down | It should simply move onto the next player if valid | Y | 0:55 |
| Selecting an invalid card | It should display that the card is invalid and allow you to select another card | Y | 1:46 |

|  |  |  |  |
| --- | --- | --- | --- |
| Draw a card | Clicking the draw button should give you a card | Y | 1:30 |
| The player picks up a card that can be placed down | If you pick up a valid card another screen should display and ask if you want to place it down or not | Y | 1:30 |
| Selecting a colour after choosing a wild card | Another screen should pop up allowing you to select a colour | Y | 3:45 |
| If there are opponents on the right and left side of the screen in two player mode | There should not be any. The opponent should be at the top of the screen. | Y | 0:55 |
| If there is an opponent at the top in three player mode | There should be none – only players on the left and right | Y | 2:17 |
| If the client and server send and receive data correctly | There should be no issues and the program should not crash at any point. | Y | 0:55 or anywhere in video |
| If all clients receive the updated game after a move is made | After a player makes a move, the server should send the updated game to all clients and their displays should update. | Y | Throughout the video |
| If the Uno button works correctly | If a user presses Uno before placing down their second-to-last card, a message should be displayed to all players telling them that the player has one card remaining. | Y | 4:45 |

|  |  |  |  |
| --- | --- | --- | --- |
| The player forgets to press the Uno button | They are dealt 4 cards and a message is displayed to all users saying that they forgot | Y | 5:35 |
| Tell the users that a player was skipped | If a skip, draw 2, reverse or wild 4 card is placed down, text should appear telling players which player was skipped. | N |  |
| Wild 4 Card | The next player’s should be skipped and they have to draw 4 cards | Y | 3:45 |
| Reverse Card | The direction of play changes so the direction image on the screen will switch and the next player is skipped. | Y |  |
| Skip Card | The next player should be skipped | Y | 10:00 or elsewhere |
| Draw 2 Card | The next player should be skipped and they draw 2 cards. | Y | 1:00 |
| The game ends | The screen should change and the winner will be displayed. | Y | 7:30 |
| The player is directed back to the main menu after the game ends | The player should return to the main menu to allow them to play again | N |  |
| Multiple games simultaneously | Two or more games should be able to run at the same time. | Y | 7:55 |
| Registering an account | The user should be able to enter their username, password and email and then register an account. | N |  |
| Logging in | The user should enter their details and if it’s correct they are given access to the game | N |  |
| Wins and Losses | The database should be updated if a player wins or loses a game | N |  |

# 5.0 Evaluation

Overall, I believe that my project was fairly successful. The game functions well and the game play proceeds smoothly with delays barely perceptible. All clients are able to connect to the server without any issues and data is successfully sent between both clients and server. Unfortunately, I did not have enough time to create my database and login system and therefore I could not implement SQL into my technical solution.

To evaluate how successful my code was, I have analysed my initial objectives to determine whether I met them and to list potential improvements. Additionally, I have mentioned the objectives that I have not met.

### 5.1 - Navigating through the menus

This was achieved successfully through the use of pygame’s event queue. After getting an event from the queue, the type of event is determined and the type represents the key that was pressed. For instance, if a user pressed the down key, the event’s type is equal to the down key. This allowed me to set a variable representing the down key to True, to inform the menu that key was pressed. The cursor rectangle would then adjust its coordinates to move downwards onto the button below.

Once it moved, the variable was reset to False to allow the user to press the down key once again. This prevents the user from holding down a key, only allowing them to tap the keys, thus preventing miss clicks. Additionally, text was displayed at the bottom of the screen to inform the users of which keys should be used to navigate through the menus.

### 5.2 - Multiple game modes and multiple games

This was successful and the game allowed you to pick between two, three and four player mode. Once a player picked their game mode, it was immediately sent to the server as an integer and the server would record the number of players that joined that game mode. Once enough players have chosen that game mode, the game would begin. If I had more time, I could have potentially included a 5 or 6 player mode.

Furthermore, multiple games were able to run at the same time since each game ran in their own thread, which meant that they would not interfere with each other in any way.

### 5.3- Reading the rules

Players were able to access the rules by pressing the Rules button on the option screen. This directed them to a webpage as stated in my initial objective. However, I could have included a button during the game to allow users to view the rules while playing, in case they needed to recap the effect of a card.

### 5.4 - Adjust the volume

This was successful through the user of a slider and buttons. When a user selected the slider, which is indicated by it being highlighted, they were able to change the volume by moving the slider left and right. There was also an option to completely mute the sound effects if they wished, by pressing the button. When disabled, a cross was displayed inside the box to make it clearly visible that the sound was disabled.

Although I intended to add music into my game, alongside the sound effects, I did not include any and this would be an improvement as it would make the game more enjoyable to play. I did not manage to find a suitable sound effect for the special cards so the sound effects only played when you placed a card down or pressed a button. A different sound was heard when an invalid card was chosen, to alert the user.

### 5.5 - Sending data back and forth

This was also successful, as clients and the server had no issues with sending and receiving data. This was possible through libraries such as socket and pickle as mentioned in the design section.

Initially, the client and server only worked on my local network meaning that players from online with different IP addresses could not play my game. This was due to my router not knowing which device to send the packets to. However, through the use of port forwarding on my router, I managed to direct the traffic to the local IP address of my laptop, which allowed users from anywhere to connect to the server hosted on my laptop. I tested this with my friend in another location and we managed to play the game successfully.

Furthermore, on the server I added time delays in between sending data and this helped to synchronise my clients and prevented the client from shutting down. If the server sent data to the client milliseconds before the client was ready to receive the data, an error occurred, so this was fixed simply by adding a short delay of less than half a second. This did not seem to have a noticeable impact on the game as data was sent and received extremely quickly as seen in the tests.

### 5.6 - Shuffle the deck and deal 7 cards

This was achieved and the deck was shuffled using the random module’s random.shuffle method. The deck was shuffled multiple times to reduce the chance of you getting multiple cards of the same colour.

The remaining cards after 7 cards were dealt to each player did form the draw pile, which can be seen when a player successfully drew a card in the testing section. The top of the discard pile was also placed in the centre when beginning the game.

### 5.7 - Viewing your cards and the opponent’s cards

As intended, all of the cards in your deck are on display at the bottom of the screen. The opponent’s cards will be placed at the left, right or top of the screen, depending on the game mode chosen. When an opponent places a card down, you are able to see that a card has been removed. Additionally, when you place a card down, the cards shuffle along to ensure that there are never any gaps in between the cards in your hand. This keeps the cards centred and prevents them from displaying outside of the window. An improvement that could be made would be to animate the cards when they are selected, so you can see them moving towards the discard pile in the centre of the screen.

### 5.8 - Placing a card down or drawing a card

This objective was also met successfully. Using your arrow keys you are able to move the cursor rectangle left and right to choose a card, and then pressing enter will select it. If the card is valid, it is sent to the server where it is placed in the discard pile and removed from your deck. If the card is not valid, text appears on the screen informing you of this, and a different sound will play. You are then allowed to select another card.  
When a wild card is chosen, you are able to choose the next colour as expected.

If you wish to draw a card, you simply hover over the draw card button and it will become highlighted. Once selected, similarly a DrawCard object is sent to the server and the execute method will be called, causing you to draw a card from the draw pile. If the card is valid, the server sends data back to you and waits to receive your decision on whether you would like to place the card down. Through the use of flags and if statements, I stopped the server from requesting the other players to choose a card until you have sent your decision back to the server.

### 5.9 - Uno functionality

This worked correctly as when a player remembers to press the Uno button before placing down their second-to-last card, it moves onto the next player successfully and informs all users that the player is on their final card. If you forget to press the button however, you will draw four cards and similarly a message will display on the screen, telling players that you forgot to press it. The button works correctly, but I did not manage to allow any player to press the Uno button to call out a player for forgetting. Instead, the program checks if the player forgot then deals them four cards. When it isn’t your turn, you cannot perform any actions so you were unable to click the Uno button to call out a player.

### 5.10 - Other functionalities

The objective of displaying the direction of play was met and an image is displayed near the discard pile to indicate whether the direction is currently clockwise or anticlockwise. This was done by loading the image and setting it’s coordinates on the screen. When the reverse card is played, the direction changes correctly to the other direction, the images are swapped and the game skips the next player as it should.

Once a player places down their final card, the game finishes and the user’s screens change to a new screen displaying the winner and a short message, along with a cheering sound effect. The window will close after a few seconds so the clients are unable to begin a new game straight away. The users must run the client once again to connect to the server and although not ideal, it is a quick process so it should not be a major concern. Letting the users return to the main menu after completion would be an improvement.