Contents

[1.0 Analysis 4](#_Toc96852821)

[1.1 Introduction 4](#_Toc96852822)

[1.11 Background 4](#_Toc96852823)

[1.12 Product Description 4](#_Toc96852824)

[1.2 Investigation 5](#_Toc96852825)

[1.21 Prospective Users 5](#_Toc96852826)

[1.22 Questionnaire 6](#_Toc96852827)

[1.23 Existing Systems 7](#_Toc96852828)

[1.24 Constraints 9](#_Toc96852829)

[1.25 Flow Diagram 10](#_Toc96852830)

[1.26 Prototypes 11](#_Toc96852831)

[1.3 Objectives 12](#_Toc96852832)

[1.4 Language/Platform 13](#_Toc96852833)

[1.41 Proposed Solution 14](#_Toc96852834)

[2.0 Design 15](#_Toc96852835)

[2.1 Description of Data Items (Data Types and Structures) 15](#_Toc96852836)

[2.2 Data Types and Structures 16](#_Toc96852837)

[2.3 – Database Design 17](#_Toc96852838)

[2.4 – UI Design / HCI Rationale 18](#_Toc96852839)

[2.41 -Login Screen 18](#_Toc96852840)

[2.42 - Register Screen 19](#_Toc96852841)

[2.43 - Main Menu 20](#_Toc96852842)

[20](#_Toc96852843)

[2.5 – Algorithms 23](#_Toc96852844)

[3.0 Technical Solution 24](#_Toc96852845)

[3.1 Login Screen 24](#_Toc96852846)

[3.2 Verification Code 27](#_Toc96852847)

[3.3 Client-Server 28](#_Toc96852848)

[3.31 Client 28](#_Toc96852849)

[3.32 Player Movement 29](#_Toc96852850)

[3.33 Network 30](#_Toc96852851)

# 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

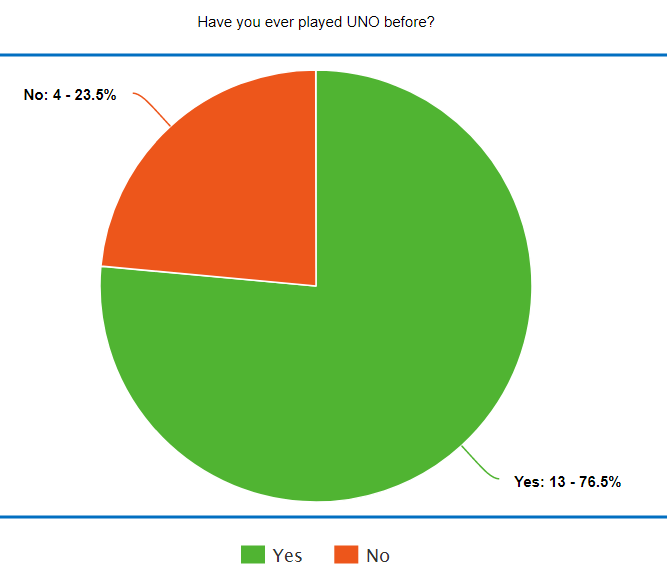
### 1.21 Prospective Users

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.

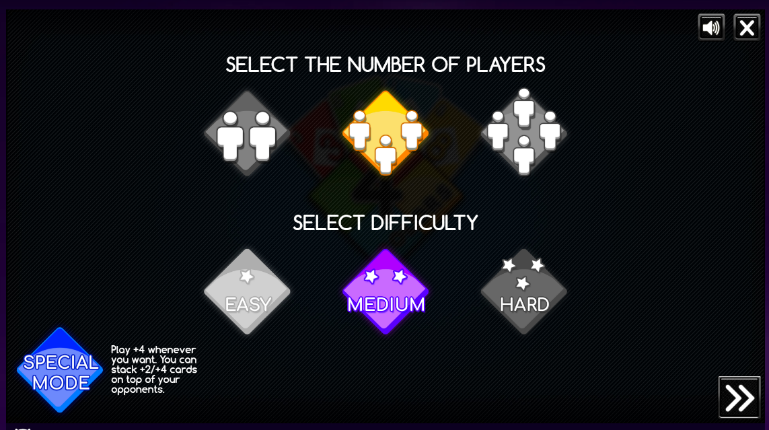
### 1.23 Existing Systems

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.

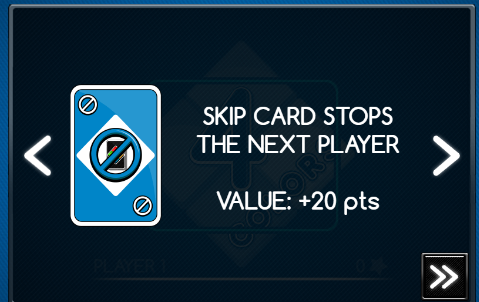


This is an online version of UNO, which is accessible via any web browser using the link <https://www.crazygames.com/game/uno-online>.

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. Similarly, my game will require fewer points to win, since 500 points are unnecessary and an excessive number of rounds will 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.

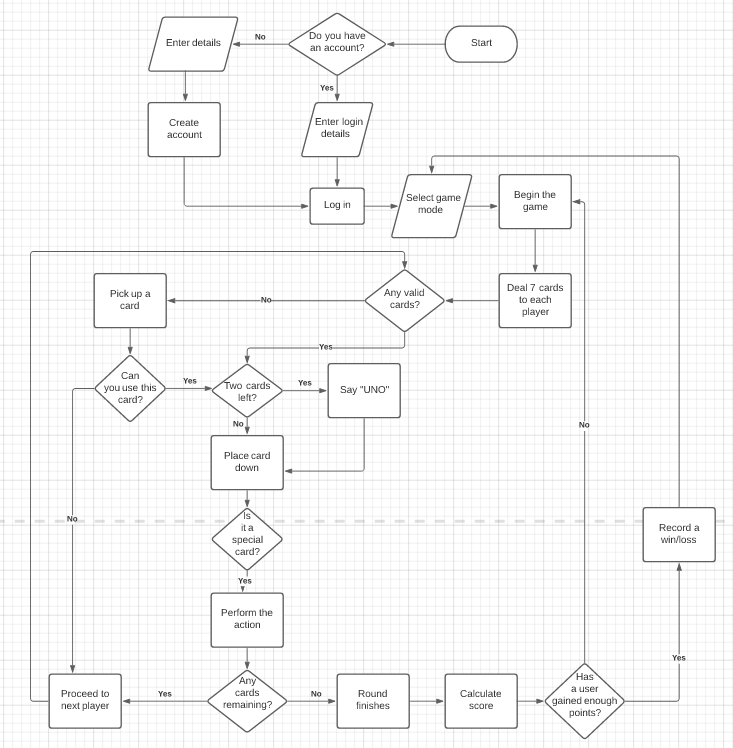
**User’s Knowledge of information technology**

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.

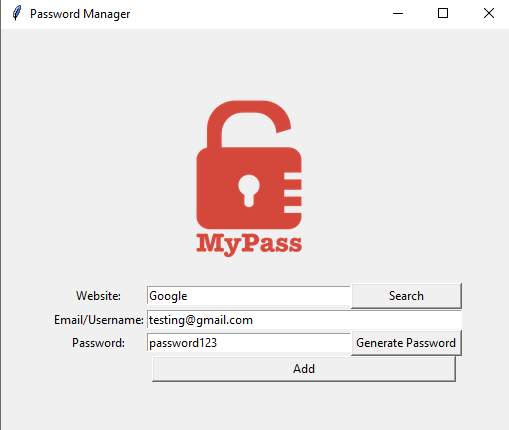
However, the system itself will be user-friendly and the necessary actions such as entering their login details and navigating through the various options should be fairly simple to complete.

### 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 user must enter a verification code sent to them by email before being allowed access to play.
* The system must store the login details inside a database and allow new users to add their login details when creating an account.
* The user must be able to pick between two/three/four-player mode and whether they want to play against the AI or another player.
* The user should be able to read the rules when needed, by clicking on a button and being redirected to a webpage.
* The system should track the number of wins and losses in each game mode by saving this to a file.
* 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 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.
* The AI will act when it is their turn.
* The AI will place down a valid card
* If they have no valid cards, they will draw a card from the deck
* 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.
* Subsequently, the scores of every player will be calculated and displayed.
* The game should be played until a final winner is determined.
* If the round finishes and a player has not gained the points necessary to win, the game will restart.
* 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 networking so another player can join the game and play against you and the AI. 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 game will be connected to a client-server database 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 them 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 Description of Data Items (Data Types and Structures)

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. |
| Verification Code | Stores the code used to verify your email | String | 10 | GkejFs81 | Not null |

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 |
| Game Mode | Stores the ID of the game mode that has been played | Integer | 1 | 2 (For 2 players) |

IPSO Diagram

|  |  |
| --- | --- |
| Inputs  -User data (Username, passwords…)  -Mouse clicks | Process  -Validation check  -Encrypt password using…  -Encrypt verification code 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 |
| Outputs  -Display the user stats (Wins, losses, number of games played, game mode)  -Output the cards onto screen | Storage  -User’s personal data  -Number of wins, losses and games played |

## 2.2 Data Types and Structures

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

* Boolean – This will be used mainly for validation, to ensure the code is running as expected.
* Floats – To store data such as the movement speed of the cards.
* Tuples – This will be used often to store RBG values to specify the colours.

Data Structures:

* Dictionaries: This will be used to link the players and their cards together. The keys in the dictionary would be the player objects and the values would be the card objects.

This allows me to manipulate the cards of different players during gameplay.

* Lists – Objects such as entry boxes and buttons may be stored in lists to allow me to loop through the list and access certain widgets.
* Queues – The cards may be stored in a queue to allow me to pop them off once the user has placed the card down.

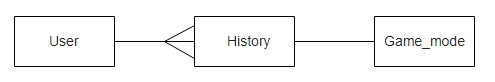
## 2.3 – Database Design

‘User’ will store the information of every user who has registered an account on the system, such as their username, password and email.

‘History’ keeps a record of the user’s past games, such as the number of wins, losses, games played and the game mode ID. A user would have multiple records because of different game modes.

‘Game\_mode’ stores data on the various game modes that the user can play. This will include the game mode ID and the name of the game mode which would be either two, three or four player. It also includes the type which would be offline or online. Offline games are played against the AI whereas the online games are played against other players.

Entity-Relationship Diagram



There is a one-to-many relationship between user and history since every user can play multiple game modes and hence have a history for the different game modes. Every record in history is for a specific game mode, thus there is a one-to-one relationship between history and game\_mode.

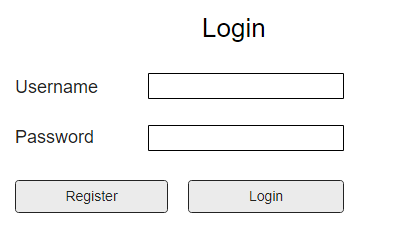
## 2.4 – UI Design / HCI Rationale

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.

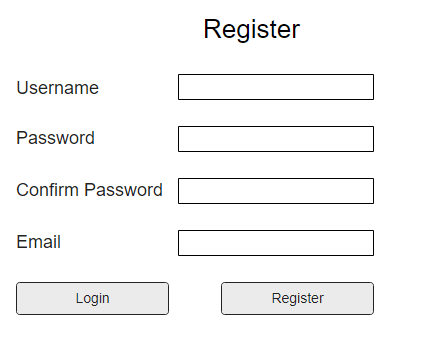
### 2.41 -Login Screen

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.

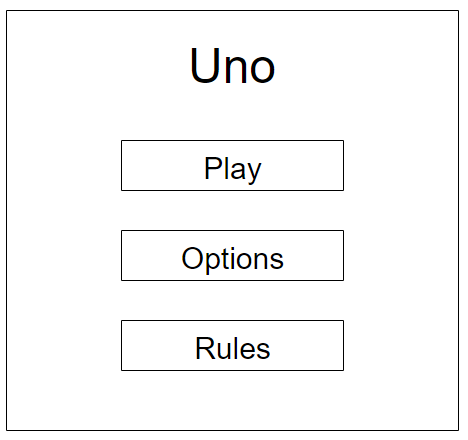


### 2.42 - Register Screen

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.



### 2.43 - 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.

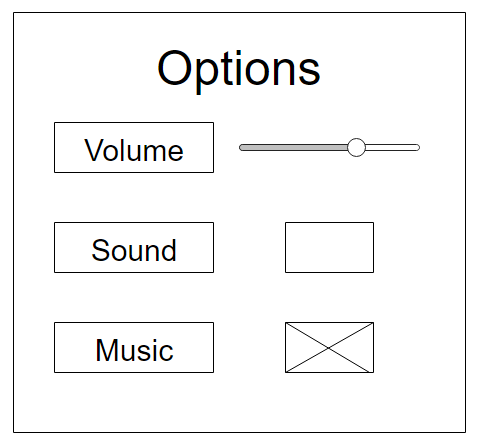
The user will be able to highlight a button using the arrow keys, and pressing Enter will select that button. To return to the previous screen, the user can press the Back button. I will add some text at the bottom of the screen to tell the user about this.

### 2.44 – 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.

### 2.45 – 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.

### 2.46 – 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.

## 

## 

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

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

## 2.5 – Algorithms

Change Colour

This algorithm will be used to highlight the input boxes that have been selected by the user.

Function change\_colour(mouseposition):  
 if rectangle position collides with mouse position:  
 rectangle.colour = activecolour  
 active = True # Used in the main program to show this box was selected

Else:  
 rectangle.colour = passivecolour # If user hasn’t clicked on this rectangle

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.

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

# 3.0 Technical Solution

## 3.1 Login Screen

import pygame

from button import Button

pygame.init()

screen = pygame.display.set\_mode((800, 600))

clock = pygame.time.Clock()

font = pygame.font.Font(None, 30)

user\_rect = Button(210, 90, 140, 32) # (x, y, width, height)

pass\_rect = Button(210, 140, 140, 32) # width becomes 200 later

email\_rect = Button(210, 190, 140, 32)

button\_list = [] # So we can loop through all buttons easily

button\_list.append(user\_rect)

button\_list.append(pass\_rect)

button\_list.append(email\_rect)

running = True

while running:

screen.fill((0, 0, 0)) # Put it before you blit so it doesn't cover the text/images

for event in pygame.event.get():

if event.type == pygame.QUIT:

pygame.quit()

mouse\_pos = pygame.mouse.get\_pos()

if event.type == pygame.MOUSEBUTTONDOWN:

for button in button\_list: # Goes through all buttons

button.change\_colour(mouse\_pos) # Colour of box will be grey until clicked on

if event.type == pygame.KEYDOWN: # Press a key

for button in button\_list:

if button.active == True: # If the button has been clicked on

button.get\_text(screen, event)

# Put labels onto screen

username = font.render("Username", True, (255, 255, 255))

screen.blit(username, (100, 100))

password = font.render("Password", True, (255, 255, 255))

screen.blit(password, (100, 150))

email = font.render("Email", True, (255, 255, 255))

screen.blit(email, (100, 200))

user\_rect.draw(screen) # To put the rectangles onto the screen

pass\_rect.draw(screen)

email\_rect.draw(screen)

pygame.display.flip()

clock.tick(60)

# Separate button File

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("lightskyblue3")

self.colour\_passive = pygame.Color("gray15")

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, surface, ev):

''' Gets your inputted text and stores it into the text variable '''

if ev.key == pygame.K\_BACKSPACE:

self.text = self.text[:-1] # Till 2nd last character

else:

self.text += ev.unicode # Unicode is the info of the button pressed.

# Adds the letter you pressed onto screen - allows you to write onto screen

def draw(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 borderwidth (2) to blit the border only

## 3.2 Verification Code

# Might remove this section later

import smtplib

my\_email = "qwerty@hotmail.com"

password = "qwerty" # My password to enter my account and send the message

with smtplib.SMTP("smtp.live.com") as connection: # Closes connection automatically

connection.starttls() # Encrypts your connection so others can't access

connection.login(user=my\_email, password=password)

connection.sendmail(from\_addr=my\_email,

to\_addrs="recipient@hotmail.com", # Recipient’s email

msg="Subject: SMTP\n\n12345678" # Subject name and then the message

)

## 3.3 Client-Server

### 3.31 Client

import pygame

from network import Network

width = 500

height = 500

win = pygame.display.set\_mode((width, height))

pygame.display.set\_caption("Client")

def redraw\_window(win, player, player2):

win.fill((255, 255, 255))

player.draw(win)

player2.draw(win)

pygame.display.update()

def main():

run = True

n = Network()

p = n.getP()

clock = pygame.time.Clock()

while run:

clock.tick(60)

p2 = n.send(p)

for event in pygame.event.get():

if event.type == pygame.QUIT:

run = False

pygame.quit()

p.move()

redraw\_window(win, p, p2)

main()

### 3.32 Player Movement

import pygame

class Player:

def \_\_init\_\_(self, x, y, width, height, colour):

self.x = x

self.y = y

self.width = width

self.height = height

self.colour = colour

self.rect = (x, y, width, height)

self.vel = 3

def draw(self, win):

pygame.draw.rect(win, self.colour, self.rect)

def move(self):

keys = pygame.key.get\_pressed()

if keys[pygame.K\_LEFT]:

self.x -= self.vel

if keys[pygame.K\_RIGHT]:

self.x += self.vel

if keys[pygame.K\_UP]:

self.y -= self.vel

if keys[pygame.K\_DOWN]:

self.y += self.vel

self.update()

def update(self):

self.rect = (self.x, self.y, self.width, self.height)

### 3.33 Network

import socket

import pickle

class Network:

def \_\_init\_\_(self):

self.client = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

self.server = "192.168.0.16"

self.port = 5555

self.addr = (self.server, self.port)

self.p = self.connect()

def getP(self):

return self.p

def connect(self):

try:

self.client.connect(self.addr)

return pickle.loads(self.client.recv(2048))

except:

pass

def send(self, data):

try:

self.client.send(pickle.dumps(data)) # dump into pickle object then send

return pickle.loads(self.client.recv(2048 \* 2)) # Getting actual object

except socket.error as e:

print(e)

n = Network()

3.34 Server

import socket

from \_thread import \*

from player import Player

import pickle

server = "192.168.0.16" # my IPv4 Address

port = 5555

s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

try:

s.bind((server, port))

except socket.error as e:

str(e)

s.listen(2) # lets multiple clients connect (2 people)

print("Waiting for a connection")

players = [Player(0, 0, 50, 50, (255, 0, 0)), Player(100, 100, 50, 50, (0, 0, 255))]

def threaded\_client(conn, player):

conn.send(pickle.dumps(players[player])) # send player object and all it's values

reply = ""

while True: # run while client still connected

try:

data = pickle.loads(conn.recv(2048)) # turn it into tuple

players[player] = data # update position

if not data:

print("Disconnected")

break

else:

if player == 1:

reply = players[0]

else:

reply = players[1]

print(f"Received: {data}")

print(f"Sending: {reply}")

conn.sendall(pickle.dumps(reply))

except:

break

print("Lost Connection")

conn.close()

currentPlayer = 0

while True: # keep looking for connections

conn, addr = s.accept() # accept any incoming connections and store it & the ip address

print(f"Connected to {addr}")

start\_new\_thread(threaded\_client, (conn, currentPlayer))

currentPlayer += 1 # Keep track of which player we're on