Name: Jaden Hamilton
Candidate Number: 1275
Centre Number: 13319
Centre Name: The Palmer Catholic Academy
Product name: CINCO!

Analysis

Introduction

I believe that card games are essential when it comes to forms of entertainment, whether that be with a large family or even just two of your close friends. They come in many different forms, including classic solitaire, Go Fish, blackjack, and one of my all-time favourites, UNO. It's a very popular, fast-paced, easy-to-play game with international recognition. However, its simplicity and reliance on luck over skill are what I think are its greatest limitations.

As a result, I have decided to create CINCO, a more engaging, strategy-driven version of UNO. It will elevate UNO's basic gameplay, introducing new rules and enhanced features, inspiring players to think carefully about their moves and outsmart their opponents. Users will be able to open the game and play by firstly logging in or creating an account if they are playing CINCO. There will be a leaderboard in the menu for players to view their total wins, total losses, and overall score. Hopefully, through CINCO, I can succeed in providing a satisfying experience for those who seek more depth than what UNO can offer.

Stakeholders

The target audience for my game is children and teenagers, as according to statistics, they are the ones who play games the most, including card games. As a result, I have selected 4 stakeholders who meet this demographic, and have created a set of questions to ask them each, and I believe the answers provided will assist me in further developing my game.

Questionnaire

- Do you enjoy playing card games online, and why?
- What makes an online card game enjoyable?
- What are some things about card games that are disadvantageous?
- What is your opinion on my card game?
- Why do you think card games are so popular?
- What makes a game engaging?

Results of Questionnaire

Stakeholder #1 - King Adomo

My first stakeholder for this project is **King Adomo**, a classmate with a strong passion for video games and card games. His experience with different gaming styles has given him an understanding of strategy and game mechanics. His feedback will be valuable for producing a game that balances strategic depth and overall enjoyment.

- **Do you enjoy playing card games online, and why?**
  "I do enjoy playing card games online because they feel worth learning the skill and concept of the game."

- **What makes an online card game enjoyable?**
  “Having different game modes/features/rules that expand the main goal of the game, because it engages people more compared to other card games locked to a single game mode.”

- **What are some things about card games that are disadvantageous?**
  “Most card games, such as Solitaire, can’t change the concept of the game. For example, Microsoft Solitaire has 5 game modes, but the concept of clearing all the cards remains the same.”

- **What is your opinion on my card game?**
  “Having an AI to play against, as well as new cards, will be useful for players who either don’t have anyone to play with or for those who want to develop their skill at the game.”

- **Why do you think card games are so popular?**
  “Card games are popular for friends/family to play as they can interact and have fun without the need for technology, as well as their simplicity, which makes them easy to learn and adapt to.”

- **What makes a game engaging?**
  “Being able to have fun while playing the game without needing an excessive level of skill.”

Stakeholder #2 - Samara Hamilton

My second stakeholder for my questionnaire is my sister, **Samara Hamilton**. She has grown up playing card games at our family gatherings, so she is familiar with the different styles of play. Recently, she has gained an interest in online gaming, so her casual and fresh gaming perspective will definitely help with the development of my own game.

- **Do you enjoy playing card games online, and why?**
  “I enjoy playing card games online because they are relaxing and easy to play.”

- **What makes an online card game enjoyable?**
  “Competing with others to win helps to make an online card game enjoyable.”

- **What are some things about card games that are disadvantageous?**
  “Some card games can easily become repetitive and boring to play.”

- **What is your opinion on my card game?**
  “I think that your card game is a nice twist on the classic game of UNO, allowing more strategy behind moves.”

- **Why do you think card games are so popular?**
  “People of all ages can play card games.”

- **What makes a game engaging?**
  “The ability to have some kind of resolve at the end of the game.”

Stakeholder #3 - Malek Gillespie-Rowe

My third stakeholder for this project will be **Malek Gillespie-Rowe**, one of my lifelong companions who has a passion for games, specifically card games. He regularly engages in gaming during youth socials at my church. Therefore, his insight into the fun factor of card games will be vital for the shaping of my own game.

- **Do you enjoy playing card games online, and why?**
  “I enjoy them because they provoke competition, you want to win, and nothing beneath that will satisfy. It is also a way to detach from the pressures of the real-life world.”

- **What makes an online card game enjoyable?**
  “Knowing you have multiple chances to win allows you to play with various people, and it's a new environment each game.”

- **What are some things about card games that are disadvantageous?**
  “It could be argued that it can become repetitive once played numerous times. Some card games require higher levels of thinking that some younger gamers may not have the patience for. Lastly, the game industry has progressed a lot over the years with video games, virtual reality; due to this, a lot of people may find card games tedious in general.”

- **What is your opinion on my card game?**
  “Brings a new anticipation, as UNO is already a competitive, fun card game. An upgrade of this would be even better. Makes the game more engaging.”

- **Why do you think card games are so popular?**
  “I feel like it's not the actual game that makes card games popular, but the atmosphere and the togetherness playing the game brings, especially with friends or family. They can be played anywhere, at any occasion, no matter the time.”

- **What makes a game engaging?**
  “The rules of the game, the difficulty (how hard is it for one person to win, the harder it is, the more value the game has), the number of players that can play at the same time, and how many unexpected turns that could happen in the game.”

Stakeholder #4 - Jamaul Días

My final stakeholder for my questionnaire is going to be **Jamaul Días.** Alongside his rooted passion for gaming overall, he is very sociable. I believe that his strong social perspective will help me ensure that the game stays engaging and enjoyable, even in a single-player format.

- **Do you enjoy playing card games online, and why?**
  “I enjoy playing card games online because it allows me to enjoy the experience of

card games even if you are unable to get the physical version."

- **What makes an online card game enjoyable?**
  "You can tailor it to your difficulty, you can play virtually anywhere, which is also really helpful."

- **What are some things about card games that are disadvantageous?**
  "Some are very time-consuming, which can be draining as people are not really having fun anymore, and they are just waiting for the game to finish. Also, if you lose a card, it can ruin the whole game."

- **What is your opinion on my card game?**
  "It looks fun and it has a spicy twist."

- **Why do you think card games are so popular?**
  "Because it can bring a community together, and it's very engaging and tactical."

- **What makes a game engaging?**
  "Thinking about what you have to do to win, and the tactical side of it."

Summary of Findings

After surveying my 4 stakeholders, I have learned that it is necessary to create a balance in my game. As I have gained insight from both sides, I have realised that variety, convenience, and competition all play a great role in online gaming. My game must have strategic depth, a high replay value, and cause my users to have a worthwhile experience. I will also avoid excessive complexity and repetitiveness within my game to prevent boredom and restlessness. With these points in mind, I realised that my original prototype needed refining. Taking into account the research from my stakeholders, I will make sure I produce an engaging form of entertainment.

Justification of new features to implement/avoid

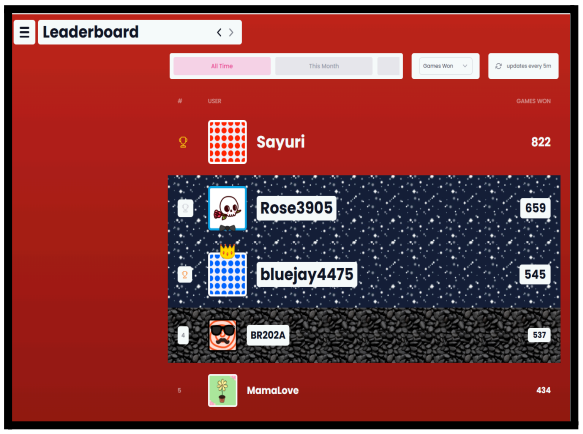
Source	Feature	Description	Justification
Malek (#3)	**Varied levels of difficulty** *"The difficulty (how hard is it for one person to win), the harder it is, the more value the game has."*	The game will vary in how easy or difficult it is to be able to complete.	This would be a great implementation to allow people of varied skill levels to enjoy the game equally. This could be done through different difficulties of an AI playing against a user, or harder game modes within the game, which are different from the original version.

Source	Feature	Description	Justification
Jamaul (#4)	**Time limits** *"Some are very time-consuming, which can be draining as people are not really having fun anymore, and they are just waiting for the game to finish."*	The game will have a set time limit, where players will only have until the given limit to play and try and win the game.	This would be very helpful as players won't be playing the game for hours trying to win, and they will be able to play multiple times to get a fuller experience of the game overall.
Own idea	**In-game shop** *"In-game shop, players would be able to buy cards to use in-game."*	The game would allow players to buy cards to use during the game. They could be normal or wild cards, and a player could use them to their advantage to win.	I don't think this concept would be suitable for my game, as the main objective of the game is to lose cards. Buying cards will increase the number of cards a user has, which would clash with the goal of the game. I also didn't think about what users would use to pay for these cards.
Own idea	**Special cards (Double cards)** *"A wild card called double cards. Instead of adding cards, a user will be able to double the card count of any other player."*	This card would allow a player to multiply anyone's card count by 2 if this wild card is placed, working similarly to a wild +4.	Although I think this card would bring a strong twist to the game, I believe that its positives are outweighed by its negatives. Cards can quickly add up to high numbers, sparking rage within a user, which is not the intention of my game.
Own idea	**Card limitation**	This will prevent players from reaching an absurd number of cards and will instead remove players from the game. I will cap this at 20.	My impression is that this will prevent players from becoming frustrated due to owning too many cards. In a game mode without a time limit, I think this would be suitable to not prolong each game played.

Research on existing products

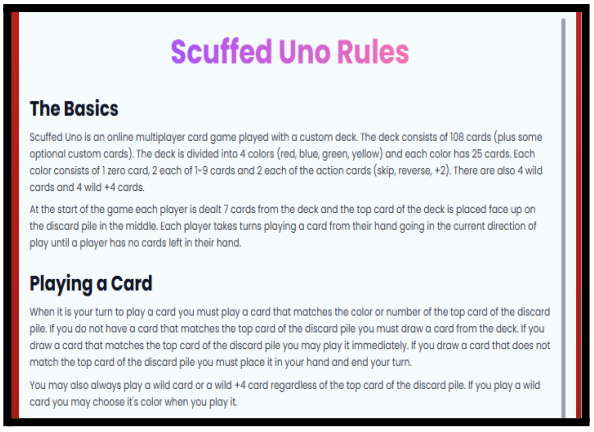
Product #1 - Scuffed UNO

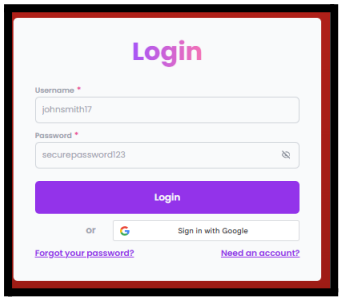
Scuffed UNO is a free multiplayer game that aims to bring UNO to online gameplay. It contains the basic rules of the game, along with a few other features that I believe will positively affect the outcome of my game.

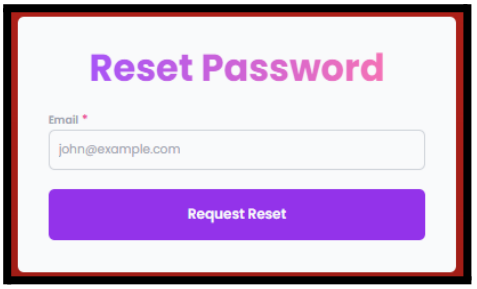


Ranking system/Leaderboard: This feature allows players to compare their overall statistics with other online players. They can change the drop-down so that they can view overall wins, total cards played, total points scored, and total games played. I think this is a very good feature to implement in my game, because players will be able to track their progress in comparison to others, which would spark challenge and competition.

Rules section: The rules section contains everything that a new user needs to know before playing ***Scuffed UNO***. It is very specific and concise, so new players will not be confused when initially playing the game. I think this is one of the most necessary features that I need to include in my game because it gives a blueprint for how to play my game and can clarify any issues that a player may have.



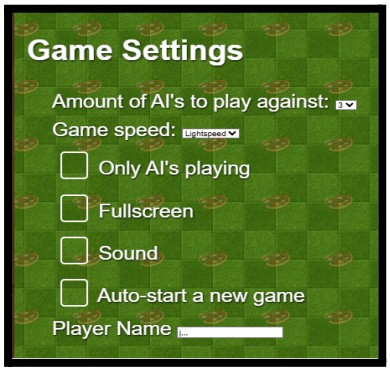




Login/Register/Account Setup: This property allows a user to create an account and sign in whenever they want to play the game. It also has options to change your password if you forget it via email or to sign in with your Google account. I really like this feature because it allows a player's data to be saved permanently and can be reloaded every single time they log in from a new platform. This is more efficient than starting the game from the beginning.

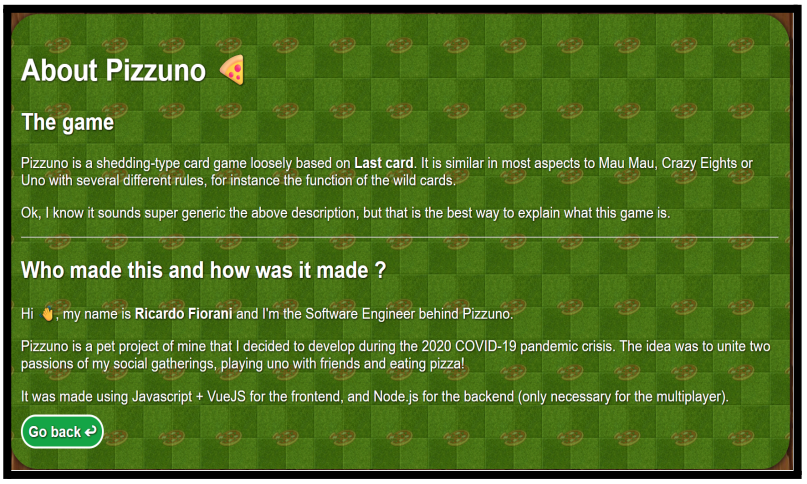
Product #2 - Pizzuno

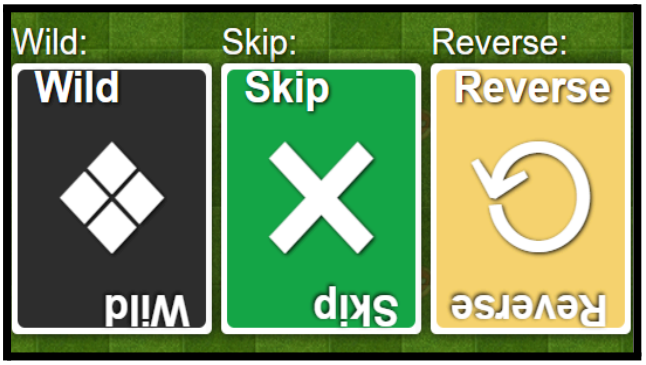
Pizzuno, like ***Scuffed UNO***, is another UNO game that has been produced virtually to play with others for free.



Game Settings: This feature allows the user to modify the settings of the game, customising their gameplay experience. For example, players can change the speed of the game, which changes the speed of each animation, making the game faster or slower depending on the user's input. I personally think this is a great attribute to the game because players can shape their own experiences and enjoy the game in their own way.

About Section: The about section of this game gives a brief description of what the game actually is. People may not be able to understand what a game is about if they just read the title or look at the game's logo. I must include this somewhere in my own game, as it would be hard to play a game, especially for casual or new players, if they don't even know the purpose of it.



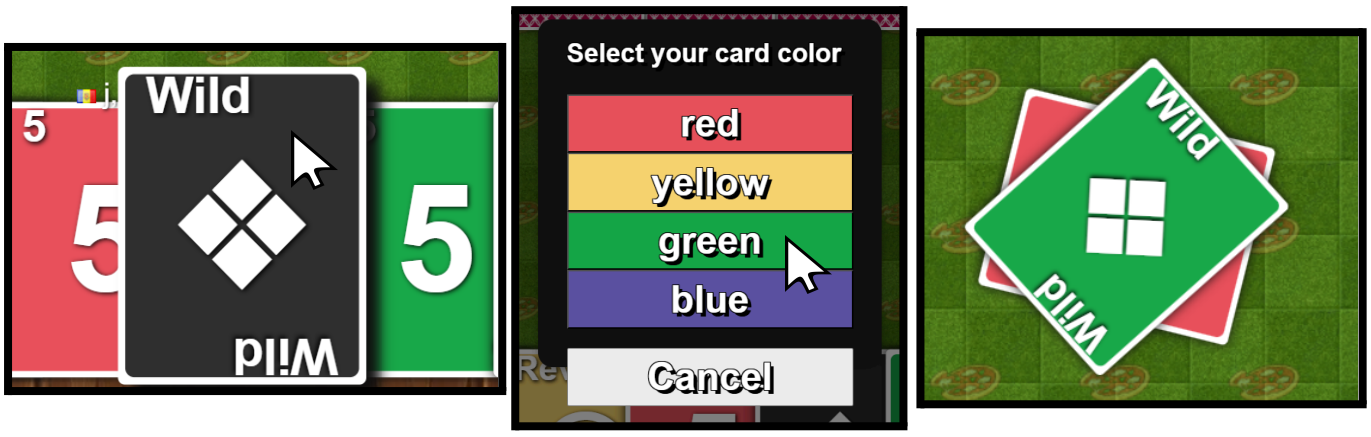


Card design: Pizzuno uses cards with simple designs to illustrate the function of each card. For example, the "Wild" card has 4 squares in the form of a diamond, representing the four colours you can use in the game. The "Skip" card has an X to show that a player cannot make a move, and the "Reverse" card has a rotating arrow to show that the order of the game has changed direction. Although I don't want design to be the main focus of my game, I believe that it will be important to find a balance between simplicity and visual appeal, like Pizzuno has achieved.

Interaction with Cards: One thing that stands out to me when playing ***Pizzuno*** is the ability to interact with different cards. This allows a user to perform different actions, which may be necessary or could enhance the gameplay of a game. This would be an effective touch to add to my game because it allows the user to interact with and engage with the game as if it were reality. Here are two examples of this:

Example 1: Here, there is a green 5 that I want to play on the deck. If I hover my cursor above the green 5, the card is enlarged to show that I am about to choose this card. The card will go back to normal when my cursor either goes onto a different card or off my set of cards completely. In this case, I hover over the red “Reverse” card.

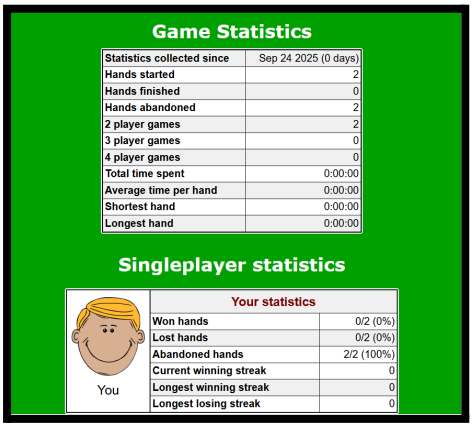
Example 2: Now I want to play my wild card next to the green 5. I hover my cursor over the wildcard. This card allows me to change the colour of the deck for me and the other players to use, which will affect the cards played. To change the colour of the card, I left-click the wild card, and a “Select your card color” GUI appears, which gives me the option of colours to choose, in this case, 4 (**Red, Yellow, Green, Blue**).



Here, I decide to choose the colour Green, which puts a green wild card on top of the deck, signifying that green is the new colour to play with.

Product #3: Crazy Eights

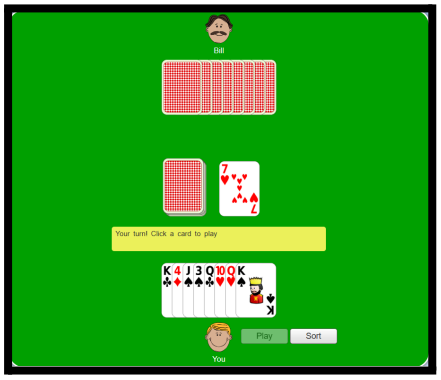
Crazy Eights is another free-to-play online game. It is very similar to UNO, in terms of the main goal and wild cards, but instead of custom cards, it uses a normal set of playing cards.



Detailed Leaderboard: In this game, there is a ranking system that has a detailed focus on your own statistics. Some of these include the day you first played, the number of games you played and won, your winning and losing streaks, and your total time spent playing. I particularly like this feature because if there is a specific detail that players aren't happy about, they can be encouraged to continue playing until they see a change in their statistics. It gives players something worth playing for.

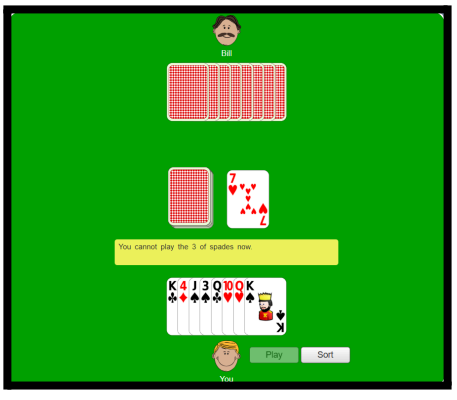
Prompts: Another key feature I have seen in this game is the prompts that are displayed. These prompts act as hints to guide a player through the game and assist with any issues a player may have. This is especially useful for casual players and people who are playing for the first time, as they may forget the rules of the game, and subtle reminders will ensure the game goes smoothly for the user.

In Example 1, the prompt is reminding the player that it is their turn to place a card.



Your turn! Click a card to play

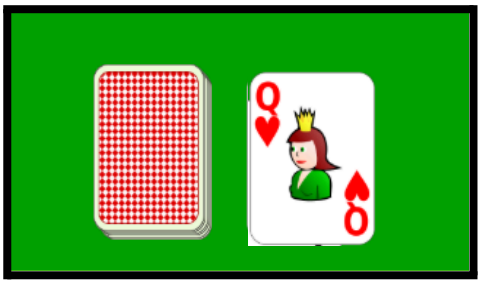
Example 2 shows that only the same number card or another card from the same suit can be played on the deck. Therefore the 3 of spades cannot be placed on the 7 of hearts.



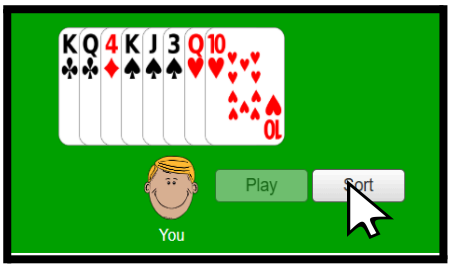
You cannot play the 3 of spades now.

Play/Sort buttons: This game also includes Play and Sort buttons for a player when dealing with their cards.

Play Button: Instead of directly clicking each card to put them on the deck, there is a Play button to clearly show which card is being chosen. In addition, this prevents players from accidentally placing a card they don't want to, and also gives them time to think about their decision. Here, I decide to play the Queen of Hearts, and this card is placed on the pile.



Sort Button: This button allows a player to reorganise their deck into their suits and the other cards within their suits. The deck can alternate between ascending and descending order. The ascending order would be ***Clubs, Diamonds, Spades, Hearts***, and the descending order would be ***Hearts, Clubs, Diamonds, Spades.*** This makes cards easier to find, especially if someone is playing under a time limit.



Findings from Research

Based on my research on these 3 products, I have found that each game has its own similarities and differences, which shape it into being an enjoyable game to play.

- Each game has its own unique features, which can all be used to benefit the experience of a user. For example, the prompts/hints in ***Crazy Eights*** can be extremely useful for learning how to play the game in real time. In addition, the ability to interact with cards in ***Pizzuno*** improves the functionality and smoothness of placing cards on the deck. Each feature has a positive impact on a user's experience.
- Each product has minimalistic designs of the cards, as well as the game's foreground and background. It is not excessive in detail, but designed enough so that players stay focused on the game at hand, without being directly distracted.
- Not all games have the ability to customise the game to the user's liking. This can be something I would include in my own game, as personalisation can offer more user satisfaction and gameplay value, compared to a default version.

Features to implement/avoid

Feature to implement	Definition & Justification	Feature to avoid	Definition & Justification
Prompts/Hints	Prompts or hints are pieces of advice or guidance given to a user to assist in gameplay. I am going to include this as I believe that my concept could appear hard to some players, so small prompts can help someone if they are struggling to play or understand my game.	**Soundtracks**	A soundtrack is a piece of audio within a game that plays throughout its duration. I am not going to include this in my own game, as this will be a big distraction from the game itself. It may throw off players when they are trying to focus on the game, appearing more as a limitation than a benefit.
Account System	Users will be able to create an account, which will store all of their statistics. They will have the ability to log into the game using their account's login details. I will also include this because most users would find it inconvenient if their statistics had not been saved, and they would like to continue from where they were at the last time they played.	**Unskippable Prompts/Hints**	Users will be forced to read the prompts, which are shown within the game. I will not put this in my own game, as I believe that this could slow down more advanced players or people who have learned the rules quickly. However, as this feature is aimed at helping all players, I will make prompts overall a customisable feature.
Customizable Game Settings	Users will be able to modify the original rules of the game. This will be a nice twist to include in my game as it will allow players to shape their own experiences without being limited to one style of play.	**Overly Complex Rules**	The rules within my game will be extremely complex. This is essential to avoid, as my game is targeted to all types of players, and should promote engagement, not frustration.

Computational methods

Thinking Ahead

Thinking Ahead is a computational method with a focus on handling a problem in the best and most efficient way possible. By doing this, a program becomes much easier and intuitive to use. An example of this is ***Preconditions***. A precondition is a requirement in a program that must be met before it can be executed. If a precondition is not met, then the program will function unintendedly or fail to function at all. Preconditions assist programmers in producing a program that runs smoothly and accurately.

I think that it is essential to add preconditions to my game because this will ensure my game doesn't cause errors when playing. An example of this is when placing a card on the deck. Preconditions will ensure that the card meets the correct criteria to be placed. If the precondition isn't in place, any card will be able to be put on the deck, which would clash with the rules of my game. I will also consider reusable program components in my program. They are commonly used functions within a program that can be stored and reused throughout the game. There are a lot of repeating functions that I will need to use throughout, such as drawing a card and placing a card, so I believe that reusable program components will positively benefit the development of my game.

Thinking Procedurally

Thinking Procedurally is another computational method that aims to break down a problem into smaller and more manageable components or procedures, ultimately making a program to understand and design. An example of this technique would be ***Problem Decomposition***. This involves breaking down a problem or task into its components. This would turn a large complex program into smaller subprograms, which are then easier to manage and eventually solve.

I believe that problem decomposition is another crucial component that I will need to include in my project because it will ensure that my coding is efficient and sequential. It will help me to effectively plan how I am going to produce my game, and will make each stage easier for me to manage and plan out. An example of this would be dealing cards at the beginning of the game. The cards have to first be in a random order and not visible to the player. A set number of unbiased cards must be given to each player of the game. There will be many other instances where I will need to break down problems for efficient production, so I will be using problem decomposition in my game.

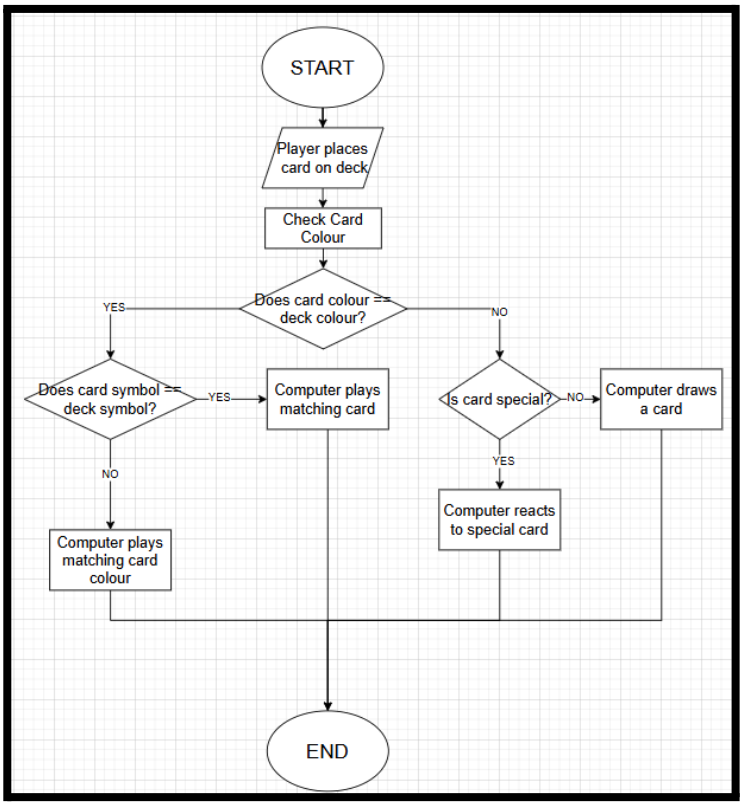
Thinking Abstractly

Thinking Abstractly, or ***Abstraction,*** is a method of computational thinking that focuses on removing excess information from a problem so that it only contains its fundamental features. Therefore, implementing abstraction will help me to reduce the time spent on producing my game, knowing that I am including the most important features, leading to efficient design.

An example of abstraction is ***Data Abstraction*****.** This is when details about how data is being stored are hidden from a user. This means that abstract data structures, such as classes, can be implemented within a program without a huge focus on the technical details. I believe that I will be using classes to create objects that I will use within my program, such as the cards, and that data abstraction will assist me with this. I won't have to focus on the unnecessary details, and I will be able to fully focus on efficiently producing my game.

Thinking Logically

Thinking Logically is a way of preparing for a range of different scenarios and gaining foresight on decisions made throughout a program. It involves choosing the best solution from a range of possibilities, leading to effective decision-making and software development. When designing my game, logical thinking will allow me to make convenient, efficient, and reasonable decisions.



An example where I may use this is when a player chooses to place a card on a deck, and this action affects the rest of the game. For example, if a user places a card on the deck, the card that the computer needs to play may change depending on the number/symbol of the card and its colour. As this is one of the main functions of the game, logic will need to be applied consistently to ensure that the game stays fair throughout.

Thinking Concurrently

Thinking Concurrently is a computational method that can complete multiple tasks at a time simultaneously and identify where concurrency can be applied in different parts of a problem. These problems are often related to one another and can therefore be done concurrently. I may use concurrent thinking when a user chooses to customise their game. For example, features such as the number of computer players at once and the number of starting cards will all need to be processed and applied at the same time. Additionally, a time limit would need to consistently run throughout the duration of a game, while cards are being dealt, drawn, and played. Concurrent thinking will ensure that the game runs seamlessly when a user is playing the game.

Performance modelling

Performance modelling is a technique used by programmers to test solutions to a problem. These methods are typically cheap, less time-consuming, and safe to implement within a software. They will allow me to gain insight into how well my program can handle different processes at once. I would use this, for example, when I am testing my computer players. Through performance modelling, I will be able to test how fast they can make informed decisions based on different scenarios within my game, and how this will affect the game's usability. These models will allow me to spot potential problems and make amendments early in my game development.

Pipelining

Pipelining is an example of concurrent processing, which improves the speed of a software's development. This is done by dividing a problem into smaller tasks to complete which can all be completed in parallel. For example, when one instruction is being executed, another instruction can be fetched and decoded. I will use this technique when collecting the total points gained from a game. I will calculate the total points earned from a game, update a player's total score, wins, and losses on the leaderboard, and finally update the total games played. As these features overlap with one another, I can use pipelining to work on different tasks at once, which reduces latency and improves the speed of updates within my game.

Visualisation

Visualisation is a computational method that uses diagrams and images to represent data in a way that is simple for humans to understand. This is a way for programmers to identify trends and patterns within a large set of data. For example, I may use visualisation to show how many and which cards are owned by each player, using simple card layouts. I could also use visualisation within my leaderboard, displaying statistics in a way that is more engaging than plain text. This will be another important feature to include in my game because it will help me to simplify complex ideas and make the game much easier to follow.

Limitations

I am certain that I am going to encounter various hurdles over the next couple of months when it comes to producing my game. Below, I have listed a few of these limitations and ways in which I can overcome them.

- **Programming skills/knowledge:** During my study of GCSE Computer Science, I was only taught Python programming at a basic level, and I know that I cannot rely on this knowledge when it comes to creating my software. Therefore, I will need to take time to learn new programming skills and techniques within Python, which will assist me in implementing the features needed for this project. One example would be learning how to use GUIs, and I can do this through tutorials from YouTube creators, such as Codemy.com and CodingWithRuss.

- **Time Management:** I am now aware that this project will take months to complete, and managing my time effectively will be a struggle. There will most likely be moments where I may fall into tiredness, frustration, and a lack of patience, impacting my motivation and overall productivity. To combat this, I will designate time slots during my weeks to focus on my game. Additionally, by taking breaks when necessary and using my time wisely and as efficiently as possible, I will be able to produce steady progress throughout the next couple of months.

- **Technicality:** I have noticed that my game is more strategy-driven compared to other card games, so I may find it challenging to keep my game fair and balance luck and skill. For example, an emphasis on strategy will risk excluding casual players, and excessive luck will limit the sense of achievement for skilled players. As a result, I will need to test different variations of my rules, observing how this affects a user's experience. Referring to my stakeholders and my peers for feedback on my prototypes will help me refine these aspects within my game, appealing to a wider audience.

- **Graphics and design:** When developing my game, I need to take into consideration its design and graphics. For example, an animation to place a card onto a deck and the file for the design of each card will take time to complete, as well as storage. To combat this, I will find a balance between simplicity and attention to detail. I will only include the number/function of each card, and its designated colour, carefully considering the total number of cards used within each game.

Requirements

My requirements will be specific software and hardware features needed for my game to be successful and function as intended.

Hardware	Justification	Software	Justification
CPU (1.6GHz +)	A CPU with a clock speed of 1.6GHz or higher will ensure that processes performed on a computer are rapid and efficient.	**Python 3.7 (+)**	Python 3.7 is the latest version, which will be able to run Tkinter().
RAM (4GB +)	A RAM of at least 4GB will guarantee that all processes within my program will be able to run smoothly and simultaneously.	**Pygame()**	Pygame is a GUI that will allow me to display my game in an engaging format, which will be able to be interacted with in-game.
Keyboard + mouse (INPUT)	A Keyboard will let a user enter their login details to sign in. A mouse will be necessary to allow a user to interact with their cards and the game deck.	**WindowsOS (10 or above)**	Windows 10 or above would be suitable for my game, as it is one of the most reliable operating systems and is therefore less likely to malfunction or cause errors when running my game.
Monitor (OUTPUT)	A monitor will allow a user to visually see what is going on in my game,	**Tkinter()**	Tkinter is a GUI that will allow me to display my menu page, which will have access to the leaderboard, settings, and account setup pages.

Success Criteria

My Success Criteria will be a way of tracking my progress throughout the production of my game. It will ultimately determine if I have been successful in what I am trying to achieve, and evaluate how effective and enjoyable my game is for my audience.

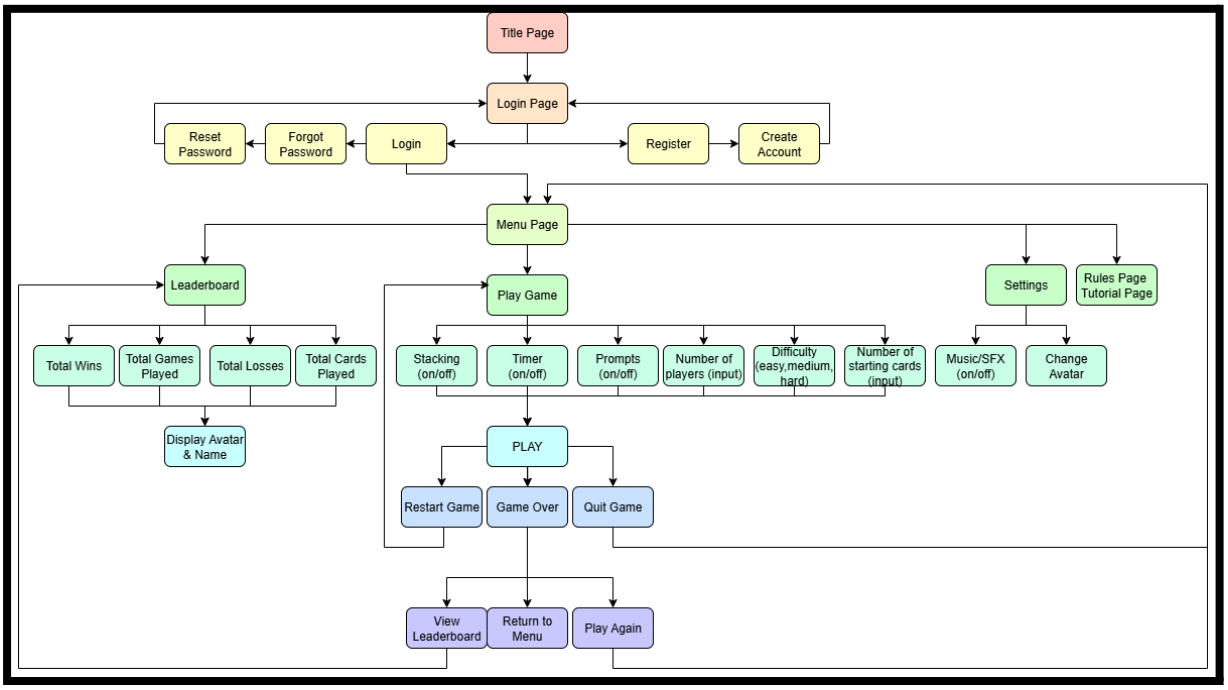
Criteria	Category	Method of Testing	Justification
Successful login method	**Usability**	*Real User Testing*	A successful login method will be necessary for virtually storing a player's data on my game, allowing them to access it any time they re-login into the program. By asking real people to test this feature, I will ensure that the login and registration process is straightforward.
Responsive controls	**Usability**	*UI Testing* *Real User Testing*	Ensuring my controls are working and are responsive is a significant factor in allowing a user to play my game. Testing this with real users will make sure that the program feels natural and runs as intended.
User-friendly menus	**Usability**	*UI Testing* *Real User Testing*	Working user-friendly menus will allow a user to navigate through different sections without difficulty, creating a smoother experience. A UI that is not easy to understand will only put users at a disadvantage when trying to play the game.
Game settings	**Usability** **Functionality**	*Logic Testing* *Real User Testing*	My game settings are the main way for players to customise their experience, and are important in making my game more enjoyable. I may test this by applying different algorithms to see how the game runs under different conditions, and asking different people to trial game settings in real scenarios.
Smooth game speed	**Robustness**	*Stress Testing*	By pushing the game to its limits by creating complex scenarios within my game, I will be able to analyse how well it can run my game at high demand, and therefore, when anything goes wrong, ensure a smooth experience throughout.

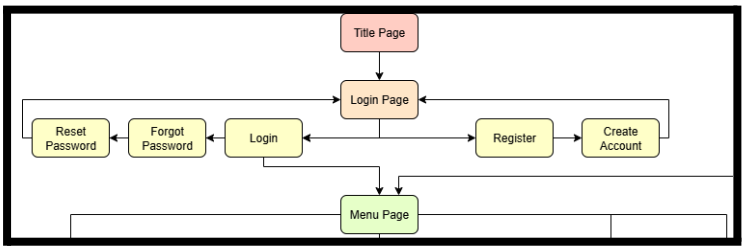
Criteria	Category	Method of Testing	Justification
Error-handling capability	**Robustness**	*Error Testing*	Error Testing will allow me to see how my game responds to invalid inputs and unexpected errors. I can test this by checking what will happen if a player breaks the rules of my game during play. My game should be able to identify that a rule has been broken. The better the error-handling capability of my game, the more likely
Clear prompts	**Functionality**	*User Testing* *Feedback Testing*	Prompts are a feature that will give pointers to users and direct them when learning how to play the game in real time. I will need to test this with other real scenarios to ensure that they remain consistent and can fulfil their purpose. I can use the feedback to polish their wording for clarity.
Fully-working new features	**Functionality**	*Feedback Testing* *Feature Testing*	My new features are what set my game across from others, so I will need to make sure these work effectively. I can test this with my stakeholders, seeing if my features run as intended, and I will be open to taking feedback on ways to improve these features.
Sound effects	**Functionality**	*Audio Testing*	Sound effects are one way of bringing a game to life, so I will need to ensure they work on different platforms and in different user environments.
Winning system	**Functionality**	*Feature Testing* *Logic Testing*	My winning system is the only way that I will be able to determine who has won and who has lost, so this section of the game must work correctly. To test this in development, I would apply all the rules required to win and create different scenarios to ensure that the winner is determined each time.

Design

Overview of Game

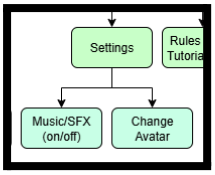
To start the production of my game, I will need to make it easier for myself to manage. I can do this by problem decomposition, where I will break down my program into smaller subprograms that are easy to understand and produce. This will help me identify the main parts of my program. I have represented this in a top-down diagram below:



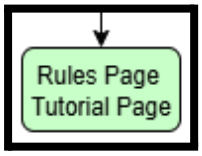


Title Page: My title page will display the name of my game, along with a login section. Users can enter their username and password to access the menu page. If they do not have an account, they will be able to register and create a new account, which will be saved to the database for future logins. If a user forgets their password, they will have the option to reset it, making sure that they can still access their account.

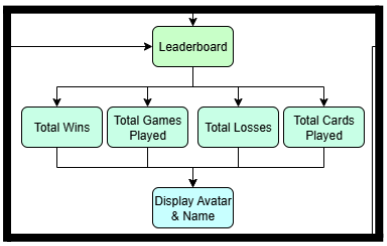
Menu Page: My menu page consists of 4 sections for a user to navigate through.



Users will have a **Settings page**. They can choose to play the game with music or sound effects and change their avatar, which will be displayed on the leaderboard.

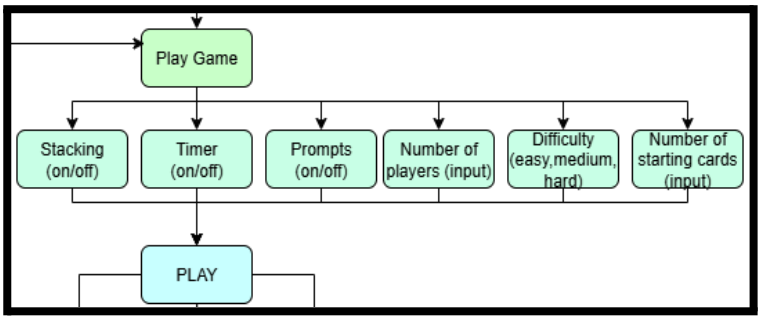


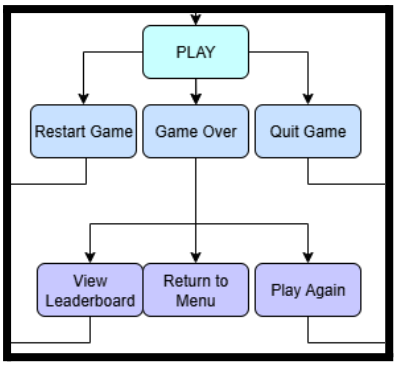
There will also be a ***Rules/Tutorial page***, which will instruct players on how to play the game. It will give a brief overview with examples, which can be applied to a user's game.



The ***Leaderboard*** menu will display the rankings and statistics of each player in the game. This will be displayed by total wins, total losses, total games played, and total cards played. Finally, there will be an option to begin ***playing the game.***

Game Settings: The game settings will be presented to the user after they choose to play the game. They will be able to enable stacking, prompts, and a timer within their game. Additionally, players will be able to choose the number of opponents, their desired difficulty level, and the number of starting cards.





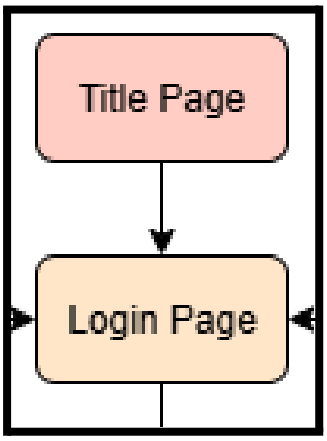
Game Play: Once a user is satisfied with the game settings they have chosen, they can begin playing the game. During gameplay, a user will have the option to restart, which will take them back to the game settings page, or quit the game, which will return them to the menu page. Once the game identifies a winner, my game will end, giving a user 3 choices: play again, view the leaderboard, or return to the menu page.

GUI

A User Interface, or UI, is a system that allows users to interact with the program. My Graphical User Interface will allow users to interact with visual things on the screen, such as menus, buttons, and icons. These will be necessary in navigating a user through my game.

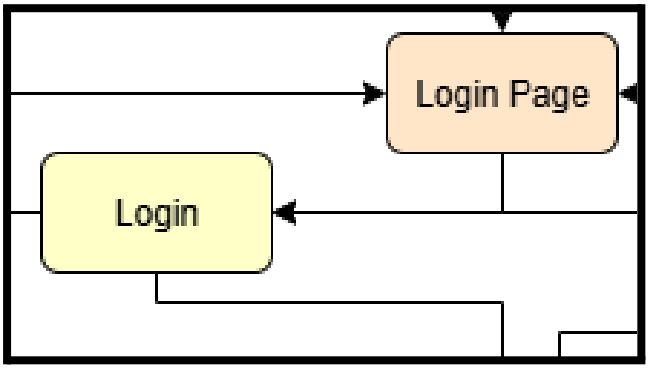
Title Page:

This screen is the first thing that will be seen when a user opens my game. I have chosen red, a strong, solid colour, to use as my primary colour. I have created a black and red background with a radial gradient. I have also chosen to include 2 of the cards I may use within the program. This will signify to my user what my game is about and also display an engaging front page.



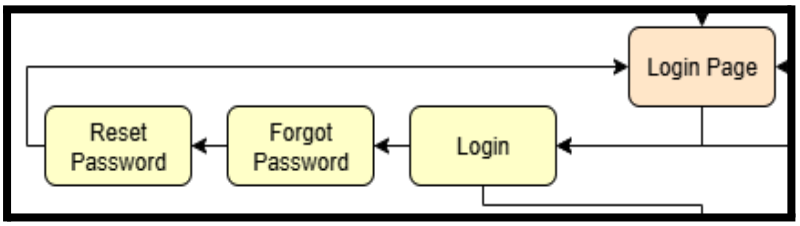
Login Section:

On my title page, users will have two choices to make. Firstly, they can log in to their account.

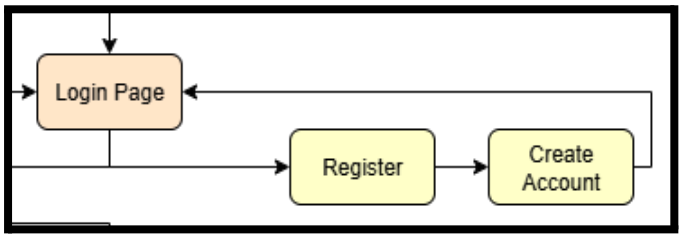


This page will require a user to enter the designated username and password associated with their account. This will give them access to the main game and allow them to play it. I have used white text with a black border, which will stand out in the red background, making it easy for users to see where to type their username, password, and login.

Forgot Password: Additionally, I have inserted the text “Forgot your password?”. This will act as an interactive button, allowing a user to reset their password if it has been forgotten. This is necessary in ensuring that a user’s data is not lost, and can be re-accessed at any time they choose to open my program.

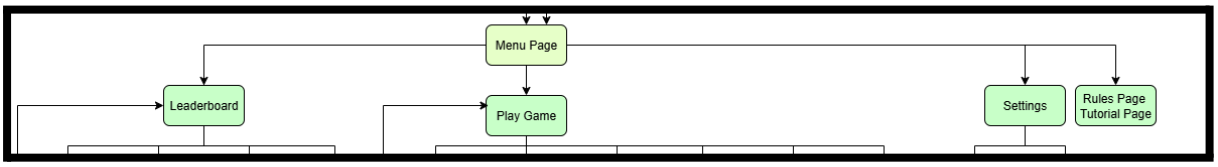


Instead of their username, users will need to enter the email address linked to their game account. This will ensure maximum security of their data, as only verified email holders will be able to reset their passwords. The design of this page will have the same visual layout as the login page, with a button that changes the user’s old password in the database, instead of logging them into their account. After this has been done, they will be directed back to the login page to use their new password.

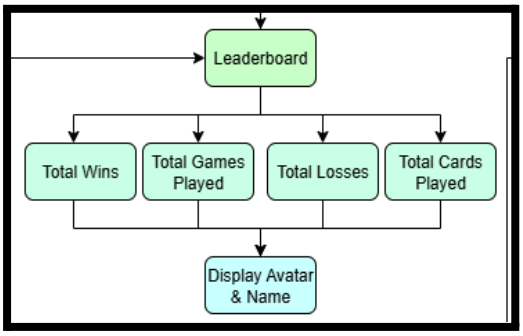


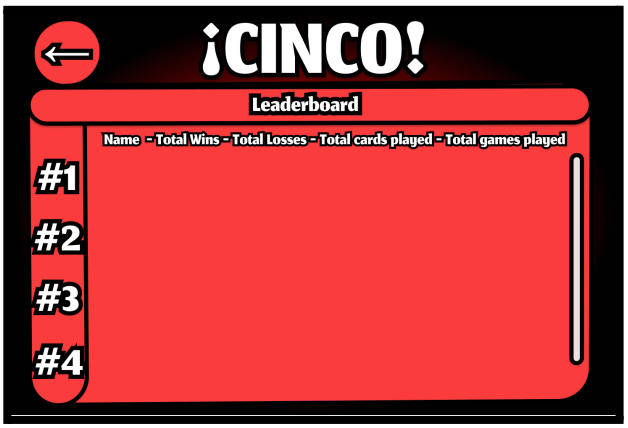
Registration: First-time users will not have an account, so they will need to register for an account on my game’s database.
Similar to the login page, users will be able to enter a username and password. Additionally, I have included an email address to verify their account, and a confirm password textbox to ensure that the password chosen is the desired password, and the chances of losing the password are reduced.

Menu Page: Here is a sample of what my menu page will look like. It contains the 4 sections from the game overview: Play, Leaderboard, Rules/Tutorial, and Settings. Each button has its own border, and the name of its function in a large text size, so that it is clear which button leads to which page. I have also included a Back button, so that users can exit the game without closing the window.

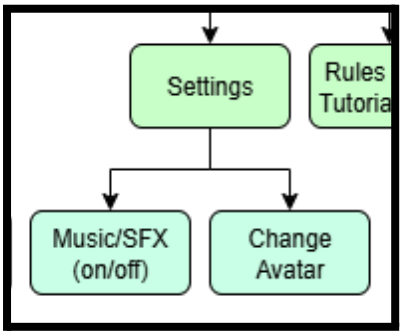


Leaderboard: I have created a rough outline of my leaderboard below:

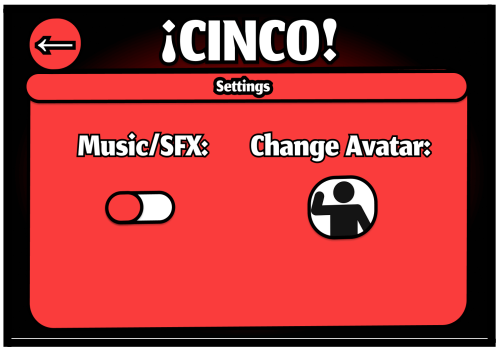




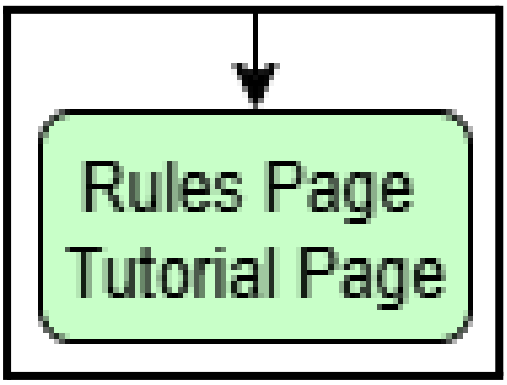
As my game has not been developed yet, I have no data to put inside the display.
However, I have separated a player's statistics, with position numbers, which start from 1 and go down to the total number of players who have played my game. Each player's statistics will be split into name, total wins, total losses, total cards played, and total games played. These sub-sections will have their own separate columns, which will make it easier for different users to see specific statistics of players. Like the menu page, there will also be a back button, so that users do not get stuck on the Leaderboard.

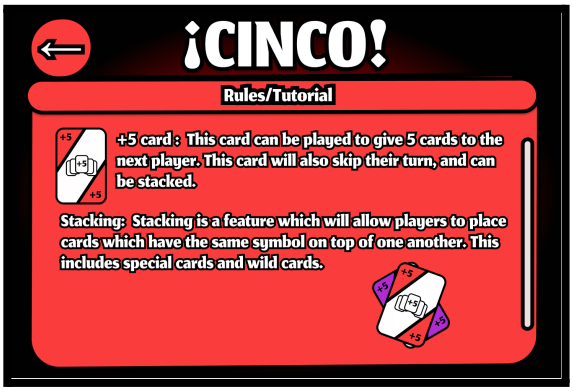


Settings: My settings page will be minimalistic, as users will only be changing settings relating to the program itself rather than the game. As this is a sample, I have only included two settings: the ability to change your avatar and mute music and sound effects. These can be changed in the menu page and during gameplay.

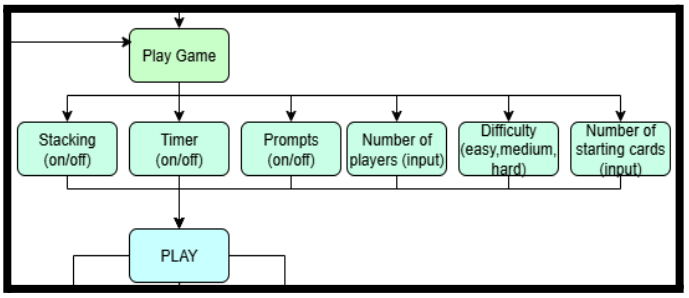


Rules/Tutorial: On this page, I aim to split the box into two parts. The first sub-section will outline all the rules of my game, and the second section will use visualisation to show an example of real game scenarios. For the rules section, I will also use visual cards to explain their functions and how to identify them within a game. I will also explain other features of the game, such as stacking, which I have done in my example. Similar to the leaderboard, I have added a back button so that users are not permanently stuck on this page.





Game Settings: When players press the “PLAY!” button on the menu page, this is a rough idea of the next page they will see. For boolean settings such as stacking, I have used toggle sliders so that they can quickly and easily be enabled or disabled. I have put another slider for the difficulty section, with the 3 options: Easy, Medium, and Hard. This allows users to choose their desired difficulty level to play with in-game. Finally, for input-based settings, such as the number of starting cards, I have put a white box for numerical values to be entered.



Usability Features

When creating my program, it will be necessary to ensure that my users can interact with the program effectively. This will provide a satisfactory experience for anyone who plays my game.

Feature	Description	Explanation	Justification
Login Button	A button that can be found on the front page of my program, letting returning users access their existing account.	When a user presses the Login button located on the front page, they will be directed to the login page. This page will require two inputs from a user: a username and a password. The database will check that the inputted credentials are valid and grant access to	Without this feature, data will never be saved when a user is playing the game. They will need to start over as statistics will be reset. This will be very inconvenient, especially for users who may play my game more than once.
Register Button	A button on the front page that allows new users to create an account.	Upon clicking this button, users will be directed to the registration page. They will be asked to enter a username, a valid email address, and a password, and will need to retype this password to confirm their choice.	Users will not be able to pass the front page without an account, so it will be vital to create one. In addition, users without an account will lose out on various core features of my game. An example of this is the Leaderboard, where their data won't be shown.
Input boxes	Interactive boxes that allow users to provide inputs by typing in text.	When necessary, a user will need to provide input for my program. They will need to click on this and use a keyboard to type in an appropriate answer that can be used.	This is important as some sections of my program cannot be completed without user input. For example, users would be unable to login or register, not passing the title page.
Forgot your Password	A link that helps users reset the password linked to	If a user forgets their password, they can click this button, where they will	Similar to the Login button, users will be unable to access

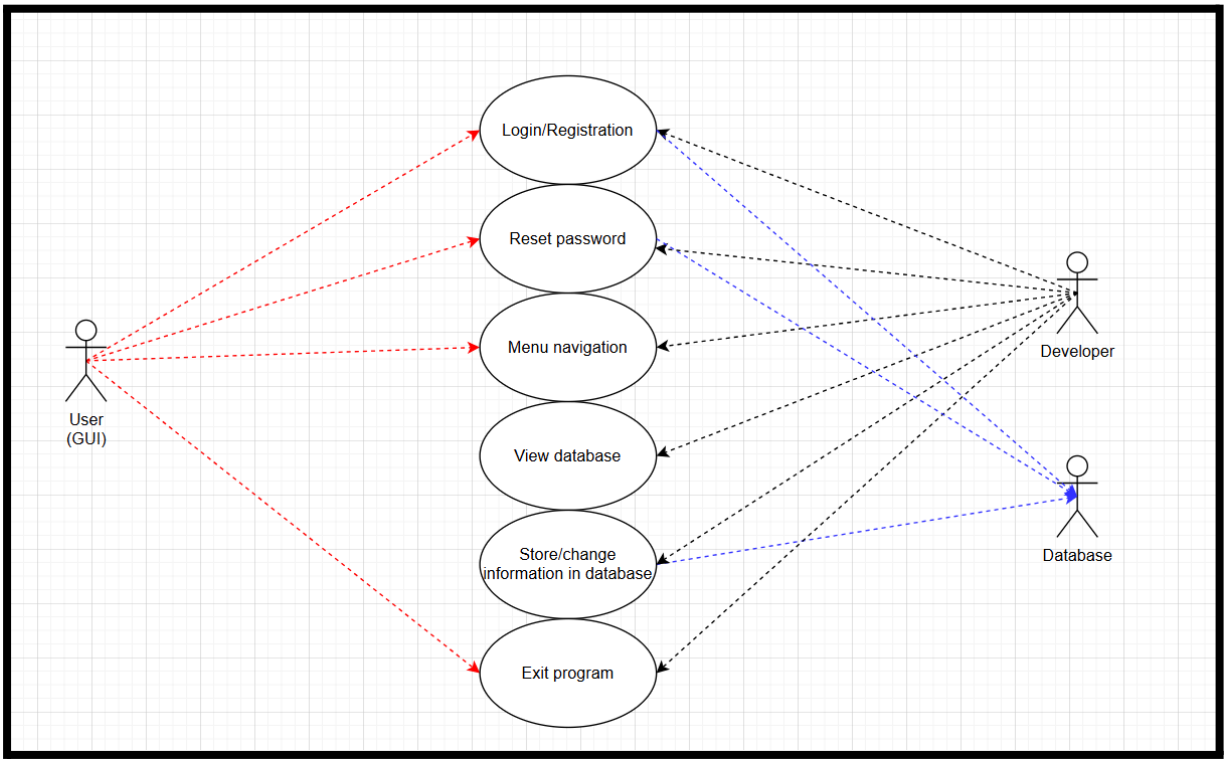
	their account.	be redirected to a page that will allow them to reset their password. They will also need to input the email address linked to their account.	their accounts if they ever forget their passwords. This button prevents this by offering an option to regain access.
Main Menu Buttons	Buttons displayed on the menu page that lead to different sections of the game.	When clicked, a user can access the desired page from the menu.	This is an essential feature to allow users to navigate through my program. Instead of being trapped on the menu page, they will be able to choose a desired page to open.
Back Button	A navigation button that returns users to the previous screen.	When this button is clicked, users will be redirected from their current page to the previous page. For example, the back button would be able to take someone from the Leaderboard back to the main menu.	This button is crucial as it will prevent getting stuck on a single page. If a user would like to go to a different page without the back button, they would need to close the program and re-open it. This is unfavourable for usability as it takes more effort.
Scrollbar	A visual tool that lets users scroll up and down a page.	When navigating through the program, if a user wishes to scroll down to read additional information on a page, the scroll bar can be utilised.	For pages with lots of text or data, it will not all fit on a user's display. A scrollbar will ensure that a user can see everything that is being presented to them.
Toggles (On/off)	Switch-like controls that allow users to enable or disable features quickly.	In my settings and game settings pages, toggles will be used for boolean options such as sound on/off. The toggle can be clicked to change its state.	By using toggles, users will be able to control their preferences much more efficiently. This will improve user customisation and accessibility.

Sliders	Adjustable bars that let users select a value within a range.	In my game settings, sliders can be used to adjust the game difficulty, ranging from easy, medium and hard. Users will be able to click and move the slider to their desired difficulty.	Sliders are important as they allow settings to be adjusted without the need for user input. It is much more straightforward to select a value in a given range using a slider.

Case Diagrams

My case diagrams are representations of what different people involved in my program, such as the developer and the user, can and cannot do. Additionally, some arrows extend from the use cases to the database to represent where data is stored. This helps to show how information flows within my game while distinguishing between different user permissions.

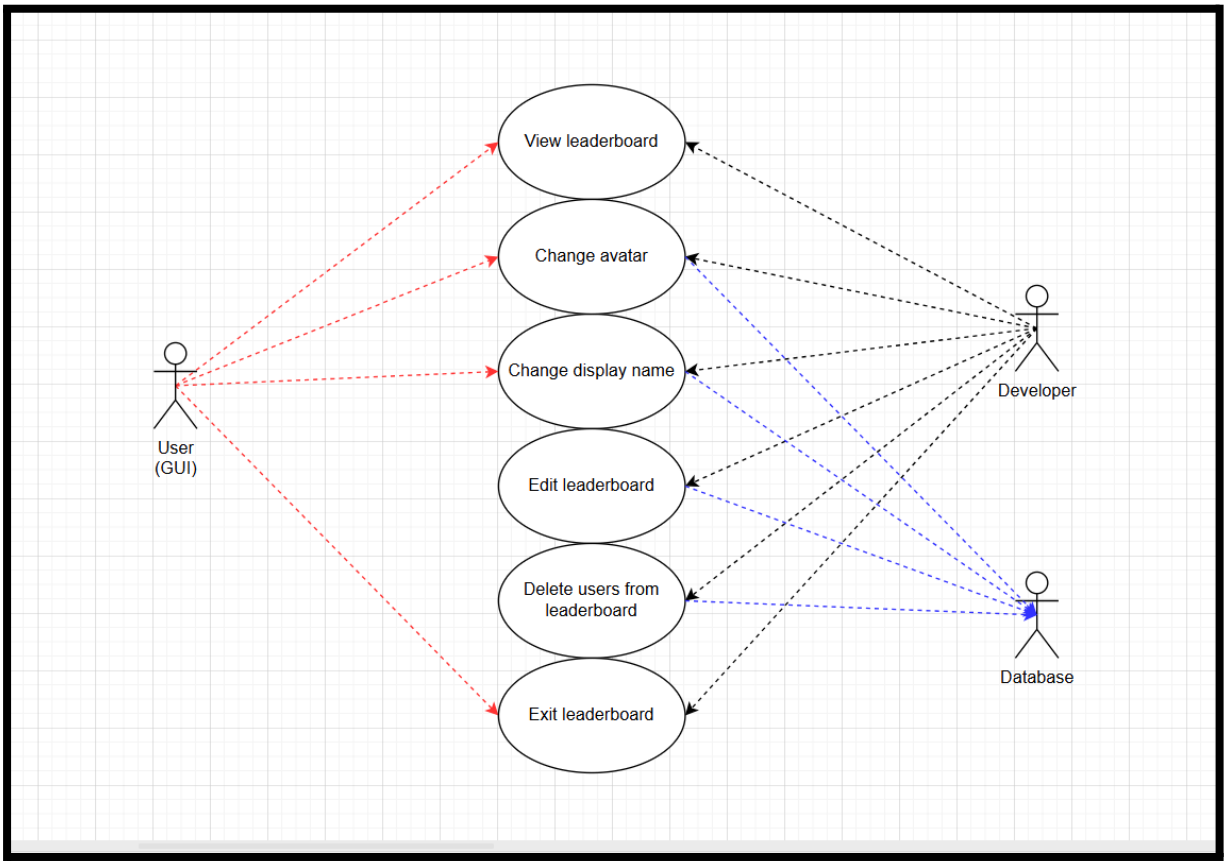
Login/Main page:



The diagram above shows who has access to what when logging in to my game and entering the main menu. As shown in the diagram, the user and the developer will be able to log in, register, and reset their password, which will all be safely stored in the database. Additionally, they will be able to navigate menus and exit the program when necessary.

However, only the developer can view the database and change information within it. This is essential because it will prevent users from interfering with the game's data, which may cause technical issues during the game's operation.

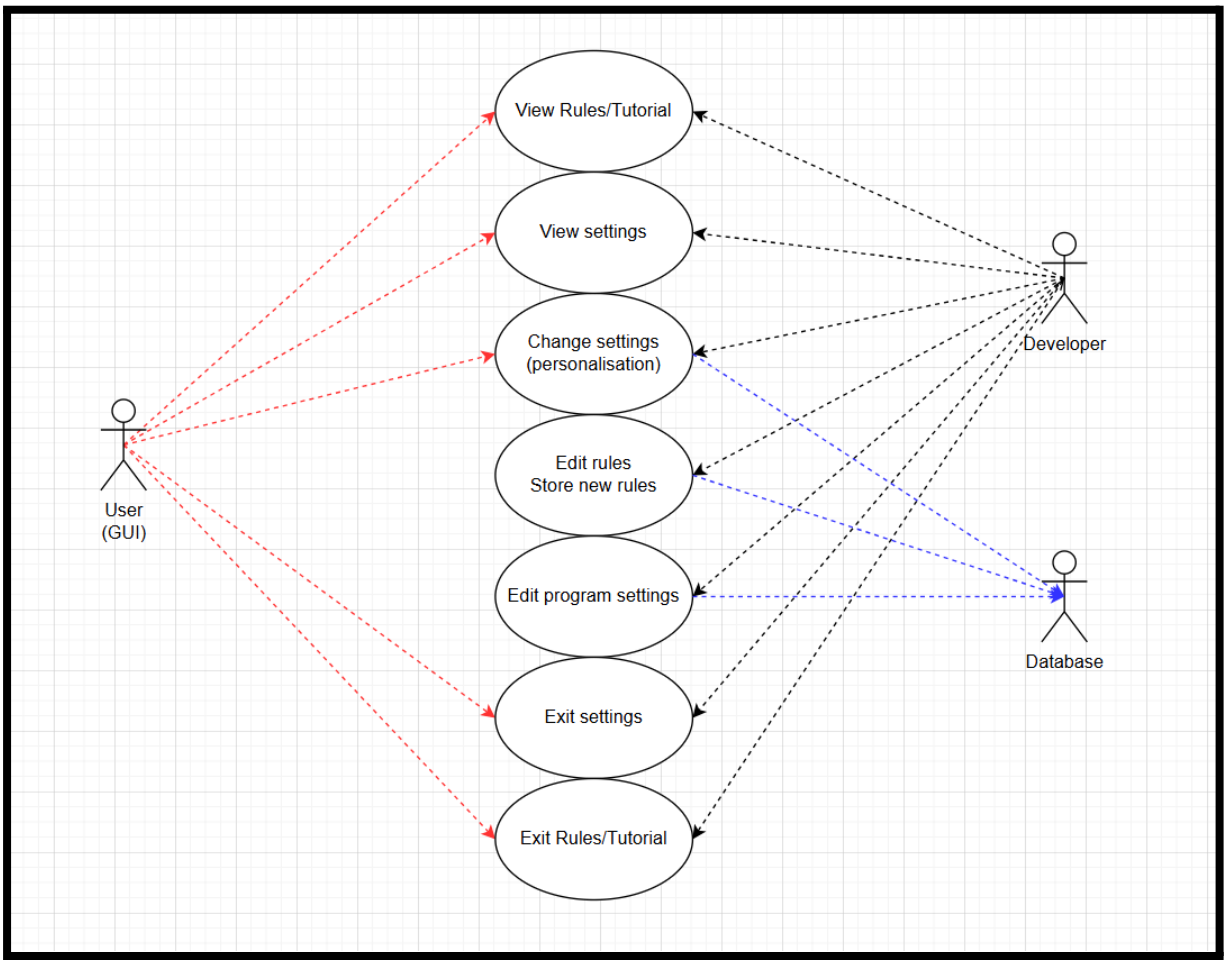
Leaderboard:



My leaderboard will also be able to be accessed by the user and developer. From the case diagram above, both will be able to view the leaderboard and return to the menu. In addition, they will be able to change their avatar and display names, which will be stored in the program's database.

In this case, the developer will also be able to edit the leaderboard and delete users from the leaderboard, which the database will save. Only the developer will be able to do this, because user access to these permissions may lead to biased scores, inaccurate rankings, and unfair representations of other players on the leaderboard.

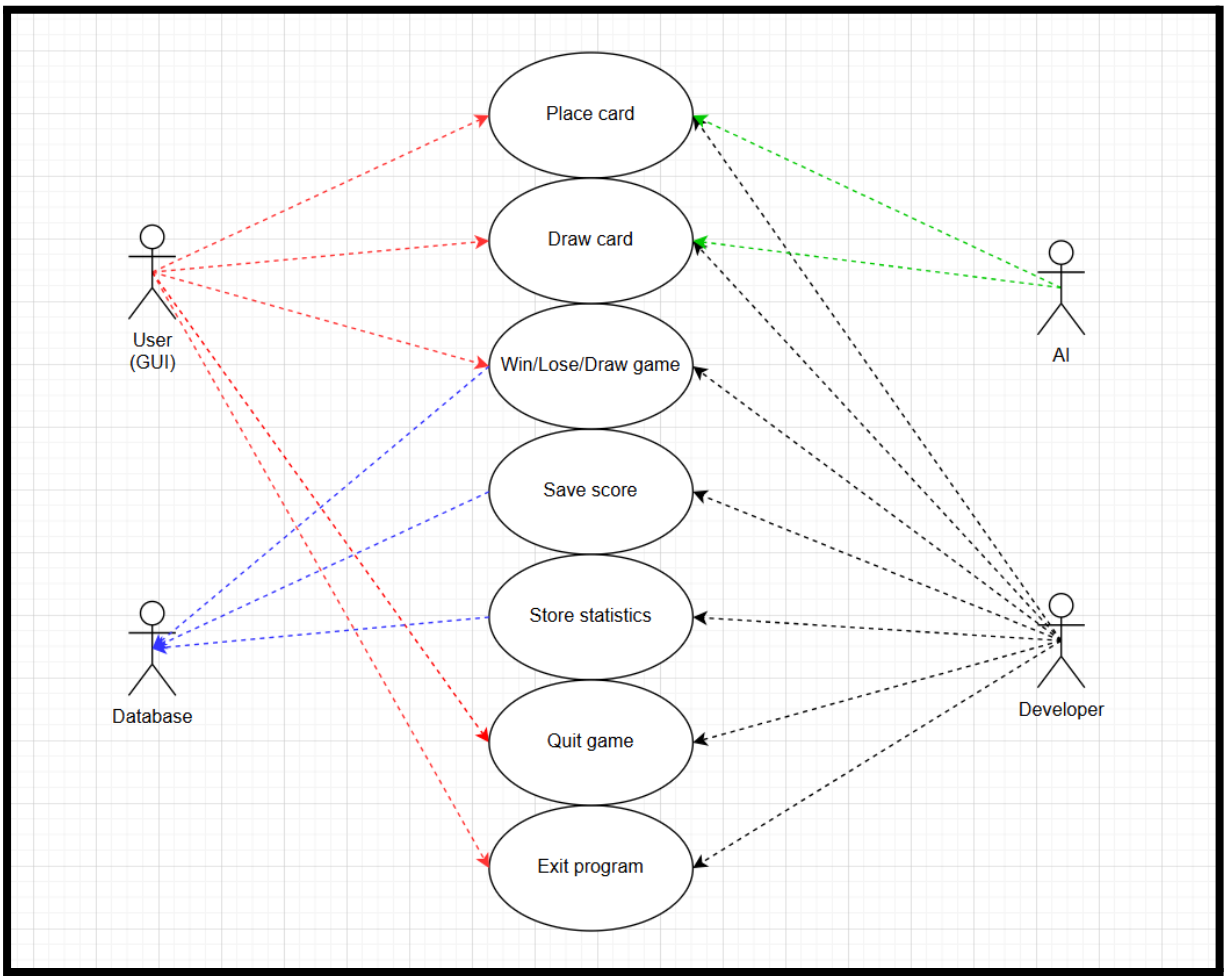
Rules/Settings:



My rules and settings will also be accessed by the user and the developer. When on this page, users will be able to view the rules and change their settings for their own personalisation, and exit both of these options when desired. This data will also be stored in the database.

The developer will also have the option to change the rules and program settings. There must be a division in permissions here because if the user were able to edit the rules and program settings, the functionality of my game would be changed. For example, a change to the rules may drastically change how players can win the game.

Game Play:



When in real gameplay, there will be four overall entities: the user, the developer, and at least one other AI player that competes against the user. Everyone will be able to place cards and draw cards in-game, the core features that determine the winner of each game. Additionally, users and the developer will be able to exit the program and quit the game to prevent them from being stuck on the game page.

However, only the developer will have the ability to save scores, edit user points, and store statistics. I have only given these permissions to the developer because users may input biased information, and this will clash with the algorithms that calculate and store user scores in the database.

Validation

In my program, there are various points where the user will need to provide an input to progress. To prevent runtime errors, I will utilise input validation to make sure that the entered data is reasonable and sufficient for my program.

Feature	Type of Validation	Explanation	Justification	Test Data & Justification
Login Username	**Presence + Length + Lookup Check**	The entered username will be checked if it has been entered in the given length (4-16 characters), and exists in the game's database.	This is necessary because it will prevent users from making a spelling error when inputting their username and ensure everyone's username is unique.	**Valid: Player1234** *Length between 4-16 characters* **Invalid: " "** *No input from the user*
Login Password	**Presence + Length + Lookup Check**	The entered password will be checked if it is within the given length (8-16 characters), and is the same password assigned to the user's account that is trying to be accessed.	This will make sure that incorrect logins are prevented when a user is trying to access their account.	**Valid: Cinco2025!** *Length between 8-16 characters, contains a special character* **Invalid: Cinco2025** *No special character*
Register Username	**Presence + Length + Lookup Check**	The username will be checked to ensure it has been typed in the correct given length (4-16 characters) and has not already been taken.	This is important as it stops someone from creating another account with the same user. This may cause logic errors in my program when storing and representing statistics.	**Valid: Cardmaster** *Length between 4-16 characters* **Invalid: Cam** *< 4 characters long*
Register Email Address	**Presence + Format Check**	The email address will be checked to ensure it has been written with an @ and a domain name like gmail.com.	These checks will ensure that a valid email address has been created for the account's contact information.	**Valid: user@gmail.com** *Contains @ symbol, valid domain extension* **Invalid: usergmail.com** *No @ symbol*

Register Password	**Presence + Length + Format Check**	The password will be checked if it has been written in the correct given length (8-16 characters), and contains at least one special character.	These checks guarantee that a user's password is secure and their account is hard to access by other users with malicious intent.	**Valid: Cinco2026!** *Length between 8-16 characters, contains a special character* **Invalid: Cinco26** *< 8 characters long*
Register Confirm Password	**Presence + Match Check**	The password will be checked to see if it has been written in the correct given length (8-16 characters), and matches the password entered in the "Register Password" section.	This is essential because both inputted passwords do not match, a logical error will be produced, and a user may not be able to access their account using the password they have created.	**Valid: Cinco2026!** *Matches the password from the previous valid example* **Invalid: Cinco2027!** *Does not match the password from the previous valid example*
Reset Password Email Address	**Presence + Format + Lookup Check**	The input will be checked if it has been written with an @, and a domain name like .gmail.com. It will also be checked to ensure the email exists in the system and belongs to a registered user.	When trying to reset a password, an incorrectly formatted email address will prevent password recovery, as the email cannot be accessed.	**Valid: me@yahoo.com** *Contains @ symbol, uses a domain name* **Invalid: me@** *No domain extension*
Reset Password New Password	**Presence + Length + Format Check**	The new password will be checked if it has been typed within the given length (8-16 characters) and contains at least one special character.	These checks are essential because they make sure that a user does not set a weak password for their account or input a blank string. It improves the security of their account.	**Valid: Reset123$** *Contains a special character, length within 8-16 characters* **Invalid: " "** *No input from the user*
Reset Password Confirm Password	**Presence + Match Check**	The input will be validated if it has been entered within the given length (8-16 characters) and matches the	This section must have the same input as the "New Password" section to confirm what the user is trying	**Valid: Reset123$** *Matches the password from the previous valid example*

		password entered in the “New Password” section.	to set as the new password for their account.	**Invalid: reset123** *Does not match the password from the previous valid example*
Difficulty Selection	**Presence + Range Check**	The selection will be validated if an option has been chosen within the given range (Easy, Medium, Hard).	If this choice is not selected, there will be a logic error, as the game will not know how difficult to make the game. Therefore, this must be checked before the game can begin.	**Valid: Easy** *Within range (Easy, Medium, Hard)* **Invalid: “ ”** *No option selected by the user*
Number of Starting Players	**Presence + Type + Range Check**	The input will be checked if it has been typed as an integer and in the given range (1-4).	This is important because without a number of starting players, the game cannot give the users AI players to play against, leading to a logic error. This will need to be a valid integer within the range.	**Valid: 4** *Within range of 1-4. Integer input* **Invalid: 0** *Outside of range 1-4*
Number of Starting Cards	**Presence + Type + Range Check**	The input will be checked to ensure it is an integer within the given range (5-7).	This is essential because a number of starting cards is required for the game to be played. Without this, no cards can be dealt. The input must be an integer within the range to prevent unplayable game scenarios.	**Valid: 5** *Within range of 5-7, integer input* **Invalid: Five** *String input instead of integer input*

Maintainability

It is crucial that I can make my program maintainable, as it will need to be repaired, updated and modified during and after development of my game. Maintainability ensures that I and other programmers can easily test and improve the program without causing major errors.

Comments

Comments are sentences within a program which describe what a section of code does, represented by a " # " in Python. This will be necessary to do during the development of my game because it will make it clear and understandable to read and edit. Additionally, if my program is edited by other programmers, they will also need to understand what is happening at each stage of my code, making comments a necessary part of my program.

Indentation

Indentation is a technique used by programmers within selections, iterations, and subprograms to visually structure code. They allow programmers to easily see which lines belong to specific blocks of code. This is important because it improves program readability, prevents logical errors, and quickly allows a programmer to identify where a block of code starts and ends. Indentation keeps the program logical and easy to debug.

Subprograms

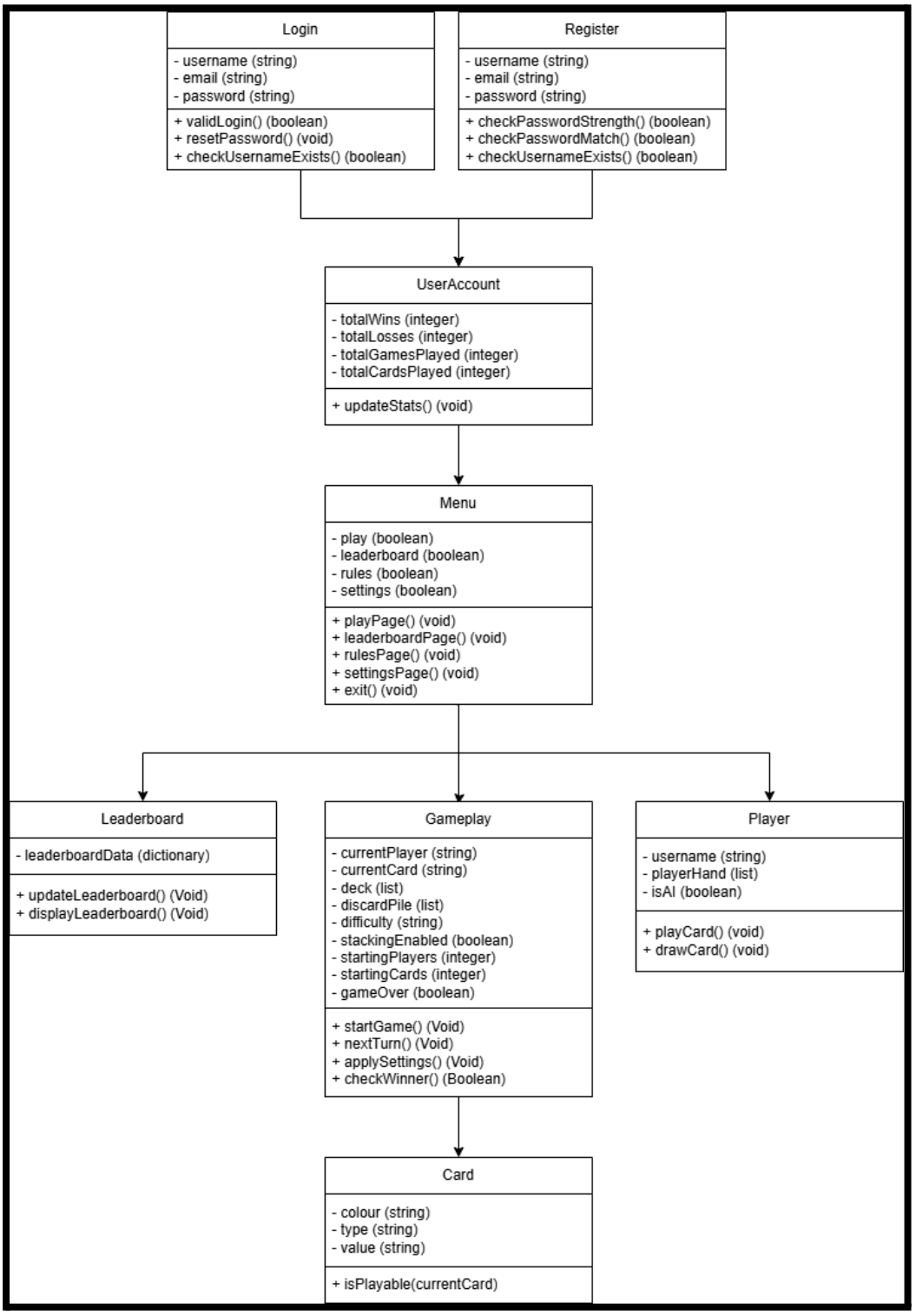
Subprograms are functions or procedures that are reusable, can be defined and can be called anywhere within my program. They make writing my code much more efficient because I will not have to rewrite similar lines of code repetitively. This would not only waste time but also waste memory space. Additionally, subprograms would allow changes to be made in one place as opposed to multiple parts of the program, improving scalability and efficiency.

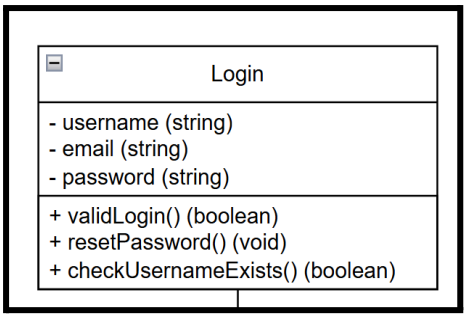
Suitable variable names

My program should have suitable variable names for its functions. This is because it will be hard to recall them when their variable name does not represent their purpose. For example, instead of calling the height of an object a vague name like "x", I can name the variable "height", as the variable's purpose becomes self-explanatory. This will especially benefit external programmers, allowing them to understand what my variable will store without having to refer to its definition.

Class diagrams

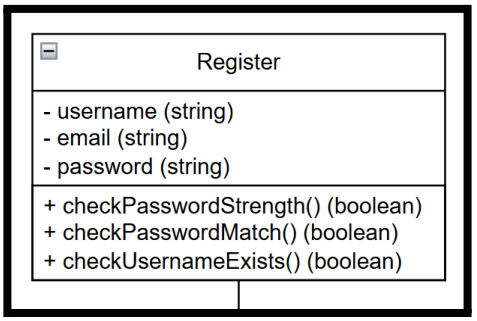
My class diagrams are visual representations of the structure of my game, including important processes, objects I will use within my game and a display of the relationships between different classes. This will help make my program more modular and maintainable over time. Here is my overall diagram below:

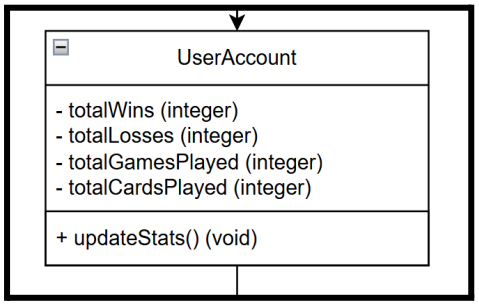




Login: The Login class is responsible for representing how a user will be able to login to my program. It contains key attributes such as username and password, which will be required in the login process, and will be encapsulated to prevent login errors. It also contains important methods, such as validLogin(), which will determine whether the password given is the correct string associated with the account trying to be accessed. This class is significant because it allows login data to be quickly and securely verified, preventing unauthorised access from others.

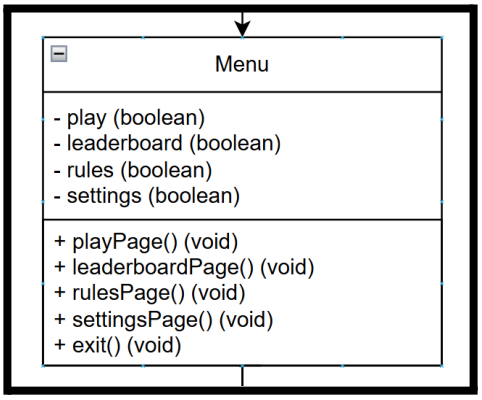
Register: The Register class will allow anyone to create an account in my program. It will contain the same attributes as the Login class, as well as different methods, such as checkPasswordStrength(), ensuring a user's password contains at least one special character for maximum security. I will need this class in my game to efficiently create accounts for users to utilise. This class will strongly link with the UserAccount class for data storage.

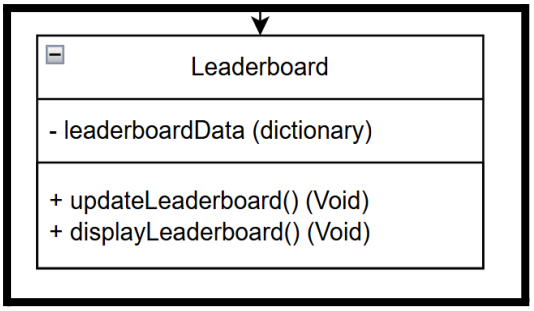




UserAccount: The UserAccount class stores data for new and existing accounts in my database. It will contain attributes such as totalWins, totalLosses, totalGamesPlayed, and totalCardsPlayed, which will aid in tracking a player's progress. Additionally, the method updateStats() will be needed to keep user statistics up to date, specifically after gameplay. This class is fundamental as player statistics can easily be updated and displayed via the leaderboard, allowing a player to track their performance over time.

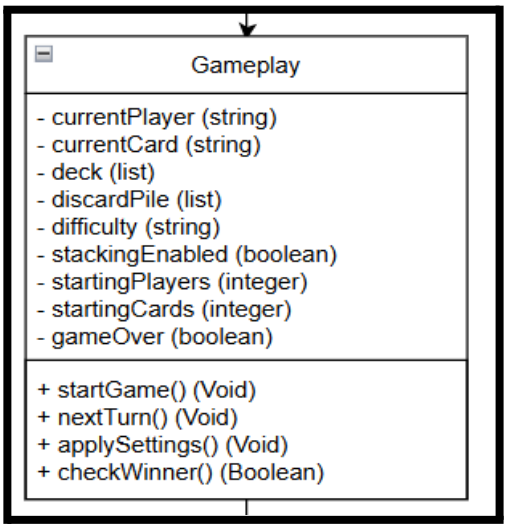
Menu: The Menu class is a class I will use to allow a user to navigate through different pages in my game, such as the play page, leaderboard page, rules page and settings page, all signified by their attributes within the class. It will also include the method exit() to allow the user to exit the page and return to the main menu page. This is a crucial class I will need to incorporate as it will help users quickly and effectively move through the game, improving the overall user experience.

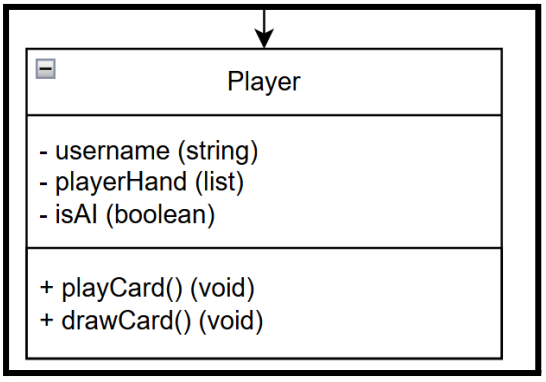




Leaderboard: The leaderboard class will be used to represent all player rankings based on performance in the game. It will contain the attribute leaderboardData, with all user statistics linked to their username, in a dictionary. Also, there will be methods like updateLeaderboard(), to update the leaderboard when data has been changed in the UserAccount page, and a change in order is needed in the ranking system. This class is needed to represent player statistics against one another, which may make my game more competitive and rewarding.

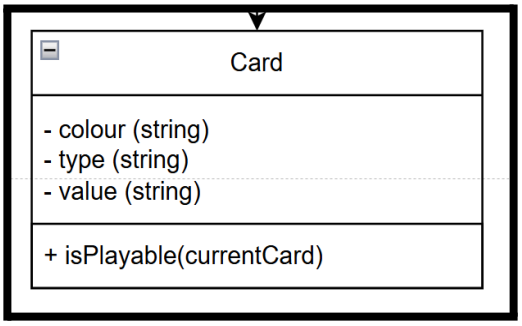
Gameplay: The gameplay class is the core of my game that controls how the game operates. It includes attributes like currentPlayer, to track whose turn it is, currentCard, to identify the current card in play, and discardPile, containing all the cards placed by all players. It also contains methods such as nextTurn(), identifying whose turn it is next, and checkWinCondition() to ensure the game eventually ends. I will need this class within my program because it enforces game rules and settings, and uses the game settings like difficulty and startingPlayers to create and adjust gameplay behaviour based on user preferences.





Player: This class will handle the user who is playing and the creation of AI players, which will play against the user in real-time. They will have their own username, playerHand, which will contain the list of cards they own during the game, and an isAI boolean to distinguish the user from the AI players. During the game, all players will be able to play a card using the playCard() method and draw a card with the drawCard() method. This class will be necessary in my game to allow players to perform actions during gameplay.

Card: This class is significant as it will allow me to create several cards, which will all be used within my game. It will contain important attributes such as their colour, card type and value so that the card can be identified easily. It will have the method isPlayable(), with the current card on top of the discard pile as a parameter. This class is crucial in enforcing legal placements and creating multiple cards for the game.



Key Variables

To keep track of everything in my game, I will need to create and assign variables to store and process data during runtime.

Variable	Data Type	Example	Purpose	Justification
username	**String**	**“Player23”**	Stores the username of a player.	This variable is important for identifying who a user is, allowing statistics to be linked to their account. A username can be formed of a range of characters, therefore, a string data type is most suitable.
password	**String**	**“Cinco2025!”**	Stores the password or a player’s account.	This variable is the main feature that allows a user to access their account. As it will contain letters or numbers, and a special character, this variable will need to be a string.
email	**String**	**“player23@gmail.com”**	Stores the email associated with a player’s account.	This variable will be necessary in allowing a user to reset their password, which will be referred back to. This variable will need to be a string data type because an email can contain letters, numbers and the @ symbol.
leaderboard	**Dictionary**	**[“Player23”: 17, 4, 23, 139 “Player24”: 12, 9, 21, 104]**	Stores the statistics of each player who has registered in my game.	It will allow data to be represented easily to show player statistics. It should be a dictionary, as each player will have different values assigned to their name.
difficulty	**String**	**“Easy”**	Stores the difficulty chosen for the game.	This variable will instruct the program on how difficult to make the game by adjusting the decision-making of AI players. This should

				have a string data type because the difficulty is represented using text values such as “Easy”.
startingCards	**Integer**	**5**	Stores the number of cards the user and AI players will use at the start of the game.	Without a number of starting cards, the game cannot be played. This variable will use an integer data type, as the starting cards are a quantity.
startingPlayers	**Integer**	**3**	Stores the number of AI players the user will play against.	As the game cannot be started with just the user playing, at least one other starting player is required. The integer is the most appropriate data type to represent this.
stackingEnabled	**Boolean**	**“True”**	Stores true or false if the user has enabled stacking in-game.	This variable is important as it determines whether the stacking rule is active in-game. It should be a boolean data type as it can only be enabled (True) or disabled (False).
timerEnabled	**Boolean**	**“False”**	Stores true or false if the user has enabled a timer in-game.	This variable will decide if a timer will be a function within the game. As this can either be enabled or disabled, this variable should be a boolean data type.
promptsEnabled	**Boolean**	**“False”**	Stores true or false if the user has enabled prompts in-game.	promptsEnabled will determine whether prompts will be shown to guide the user during gameplay. The user can choose to enable or disable this function, making a boolean data type the most ideal.
soundEnabled	**Boolean**	**“True”**	Stores true or false if the user has enabled sound effects	This variable decides whether the user will hear sound effects

			in-game.	when playing the game. This should be a boolean data type because the user can disable or enable this within the settings page.
playerHand	**Array**	**[“Red 5”,“Wild +5”]**	Stores the list of cards that a player currently has and can play in-game.	This variable will allow cards to be added and removed from a player when cards are placed or drawn. As multiple strings could be owned by a player, an array is the most appropriate data type to use.
deck	**Array**	**[“Blue reverse”,“Red 8”]**	Stores the list of cards that are within the main deck.	This variable is important because it decides what card a player will get when they must draw cards. An array is appropriate because it allows cards to be easily accessed and removed as the game progresses.
discardPile	**Array**	**[“Green 0”,“Yellow 7”]**	Stores the list of cards that have been played by each player in the game.	This variable is needed to keep track of which cards have been placed by the players. It should be an array because it allows cards to be quickly added when a card is placed by a player.
currentCard	**String**	**“Blue skip”**	Stores the card that is currently on the top of the deck.	This variable is important as it will determine the next move any player can make. It uses a string data type to store the colour of the card, as well as its type.
currentPlayer	**String**	**“Player23”**	Stores the name of the player whose turn it currently is.	During gameplay, currentPlayer will make tracking the game much easier. As this variable stores the name of a player, it

				must have the string data type.
validMove	**Boolean**	**“False”**	Stores true or false if a player has made a correct move according to the rules.	This variable will prevent any player from making an impossible move during gameplay. It will use a boolean data type because each move will either be possible or impossible.
gameOver	**Boolean**	**“True”**	Stores true or false if any player has won the game.	This variable will prevent the game from continuing if at least one winning condition is met. It will use a boolean data type because there are only two possibilities: a winner or no winner.
winner	**String**	**“Player24”**	Stores the name of the player who has won the game.	This variable is necessary for displaying the winner of each game, making statistics easier to update. As the username of a player will be saved, this variable must use a string data type.
totalWins	**Integer**	**24**	Stores the total number of wins a player has accumulated.	This variable allows the program to record a player’s performance, displaying their success on the leaderboard. It will use an integer data type to represent the total as a numerical value.
totalLosses	**Integer**	**8**	Stores the total number of losses a player has accumulated.	This variable will count each loss gained by a player, so that it can be stored on the leaderboard correctly. An integer data type is used, as this numerical value will increase every time a player loses a game.

totalGamesPlayed	**Integer**	**32**	Stores the total number of games a player has played.	This variable is needed to track how many full games a player has played for an accurate representation. An integer data type will be used as it reflects a countable number of games played.
totalCardsPlayed	**Integer**	**177**	Stores the total number of cards a player has played.	This variable is important in counting the number of cards played by a user overall, to then be shown in the leaderboard. Similar to totalGamesPlayed, the total reflects a countable number of cards placed, therefore, it will need to have an integer data type.

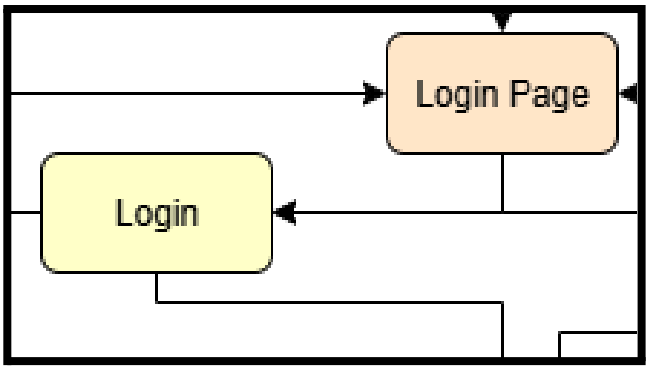
Algorithms

My algorithms will describe key processes in my program. Each algorithm will be written in pseudocode to show the logical structure of the game before they are implemented. Writing these algorithms beforehand will ensure that my program is easy to understand, debug and can be used effectively in Python.

Login:

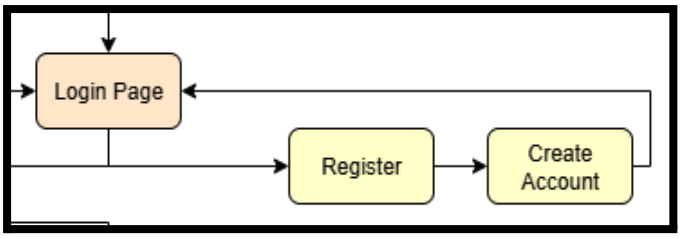
```
INPUT Username
INPUT Password

GET password_db
GET username_db

IF Username == "" OR Password == "" THEN
   OUTPUT "Not all fields are filled in - Please try again"

ELSE
   IF username EXISTS IN username_db THEN
      IF password == password_db[username] THEN
         OUTPUT "Login successful!"
         menuPage()

      ELSE
         OUTPUT ("Invalid password entered - Please try again")
      ENDIF

   ELSE
      OUTPUT "Invalid username entered - Please try again"

   END IF
END IF
```

My Login algorithm will be responsible for validating a user's access to the main program. Firstly, a presence check is performed to ensure that an input has been typed by the user. Then the username and password will be checked to see if they exist within my program's database. This algorithm will ensure that only users with an account can access the game, preventing unauthorised access from external users, making an account is a requirement when using my program.



Register:

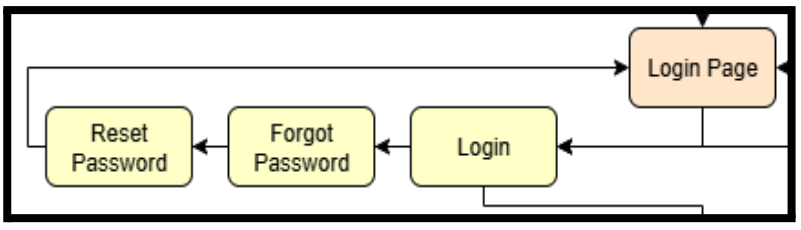
```
INPUT Username
INPUT Email
INPUT Password
INPUT Confirm_password

GET username_db
GET password_db
GET email_db

IF Password == "" OR Username == "" OR Email == "" OR Confirm_password == "" THEN
   OUTPUT "Not all fields are filled in - please try again"

ELIF Email IN email_db OR username IN username_db THEN
   OUTPUT "Already taken - please try again"

ELIF "@" NOT IN Email OR "." NOT IN Email THEN
   OUTPUT "Invalid email format - please try again"

ELIF len(Password) < 8 OR len(Password) > 16 THEN
   OUTPUT "Enter a password in the range of 8-16 characters"

ELIF password HAS NO special character THEN
   OUTPUT "Weak password entered - please enter a special character"

ELIF Password != Confirm_password THEN
   OUTPUT "Passwords do not match - please try again"

ELSE
   ADD Username, Email and Password TO user_details_db
   OUTPUT "Account created successfully"
   titlePage()

ENDIF
```



This algorithm will be used to allow new users to register an account to be saved in the database for future logins. To start, a presence check is performed to check that all inputs have been filled in. Next, the username and email are checked to see if they exist already in the database, requiring a user to choose a different email address and username. Then, a range of validation checks, such as format and length checks, are performed on the email address and password, as well as a match check between the password and the confirmed password, to ensure they can be validated for usage in my program. Finally, the username, email, and password are stored in the database to be accessed again when the user tries to log into the game. The Register algorithm will ensure accounts are created correctly and stored successfully for access.

ResetPassword:

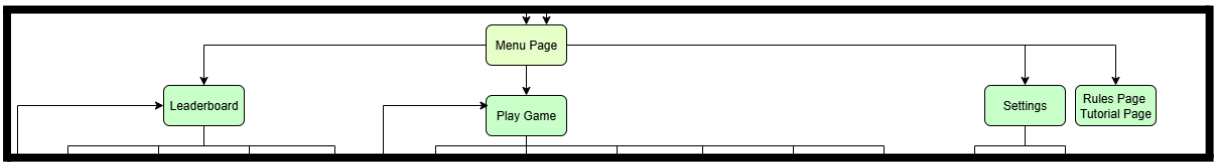
```
INPUT Email
GET email_db

IF Email == "" THEN
   OUTPUT "No email entered - please try again"

ELIF NO "@" IN Email OR NO "." IN Email THEN
   OUTPUT "Invalid email format - please try again"

ELIF Email NOT IN email_db THEN
   OUTPUT "Email not found - please try another email"
ELSE
   INPUT New_password
   INPUT Confirm_password

   IF New_password == "" OR Confirm_password == THEN
      OUTPUT "Not all fields are filled in - please try again"

   ELIF len(New_password) < 8 OR len (New_password) > 16 THEN
      OUTPUT "Enter a password in the range of 8-16 characters"

   ELIF New_password HAS NO special characters THEN
      OUTPUT "Weak password entered - please try again"

   ELIF New_password != Confirm_password THEN
      OUTPUT "Passwords do not match - please try again"

   ELSE
      UPDATE password_db[email] WITH New_password
      OUTPUT "Password reset successfully"
      loginPage()

   END IF
END IF
```



The ResetPassword algorithm enables a user to recover their account and access if their password is forgotten. It firstly validates the input of a verified email address in the database, and then applies validation checks on the password, such as a presence check and a length check, to ensure that the new password is strong enough for account security. Once the new password is validated by the algorithm, this password replaces the old one found in the database. This algorithm is crucial for sustaining user accessibility to their account, reducing the probability of users being permanently locked out of their accounts.

Menu:

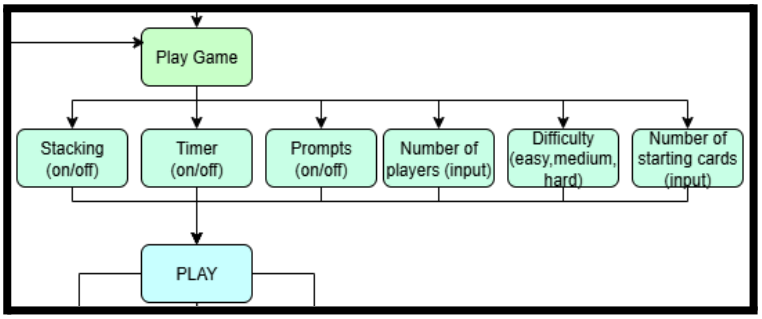
```
IF play_button IS clicked THEN
    CALL playPage()

ELIF rules_button IS clicked THEN
    CALL rulesPage()

ELIF leaderboard_button IS clicked THEN
    CALL leaderboardPage()

ELIF settings_button IS clicked THEN
    CALL settingsPage()

ELIF exit_button THEN
    CALL exitGame()

END IF
```



My menu algorithm will be a simple navigation system, allowing users to choose between different pages in my game, such as the leaderboard page and the settings page. When each button is clicked from the menu, the corresponding page will be called, and the subroutine will be executed. Additionally, users can exit the game by clicking the exit button, which will be located on the menu. This will ensure that users are not stuck on the game itself and can exit whenever they choose.

SetupGame:

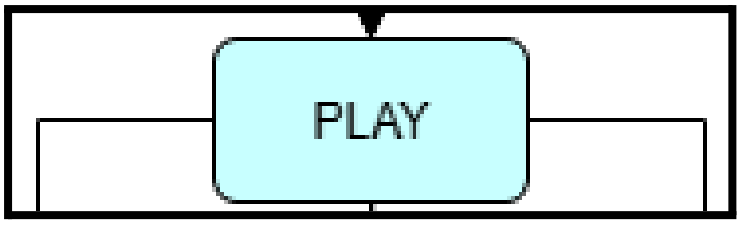
```
CALL applySettings()

INITIALISE deck WITH all card objects
SHUFFLE deck

FOR EACH player IN players
   DEAL startingCards FROM deck TO playerHand
END FOR

SET currentCard TO top card of deck
MOVE currentCard TO discardPile
SET currentPlayer TO players[0]

OUTPUT "The game has started!"
```



The SetupGame algorithm is a simple structure that will set up each game before it is played. Firstly, the applySettings() method is called to signify which settings have been enabled and disabled by the users in the game settings. An example of where this will be applied is in the for loop, which deals the number of cards chosen by the user from the game settings to each player. Finally, the starting player is chosen, and the current card is assigned. An output statement is shown to highlight the start of the game after setup. This algorithm will be essential as it will show which cards each player will have, and what cards can be played based on the current card, linking to the PlayTurn algorithm below.

PlayTurn:

```
SET playableCards TO [ ]

FOR EACH card IN currentPlayer.playerHand
   IF card IS playable ON currentCard THEN
      ADD card to playableCards
   END IF
END FOR

IF playableCards == [ ] THEN
   CALL drawCard(currentPlayer)
   OUTPUT "No cards could be played,", currentPlayer, "has drawn a card."

ELSE
   SHOW playableCards
   INPUT selectedCard

   IF selectedCard IS playable ON currentCard THEN
      MOVE selectedCard to discardPile
      REMOVE selectedCard FROM currentPlayer.playerHand
      SET currentCard TO selectedCard
      OUTPUT currentplayer, "played", selectedCard

   ELSE
      OUTPUT "Invalid move - please try again"

   END IF
END IF

CALL checkWinCondition()
```



The PlayTurn algorithm will handle gameplay logic for each player's turn. It starts by determining the cards that can be played based on the cards in the player's hand and the current card on top of the deck. If no cards are found in this list, the player must draw a new card. If the player chooses to play a card that is in the playableCards list, the move is validated, and the discard pile and current card are updated accordingly. However, if the card chosen does not appear in the playableCards list, the player is notified that the move is invalid and a different card is required. Finally, the checkWinCondition() function is called, which will check if any player has won after playing a card. This algorithm is important as it creates an effective system for players to place cards and validate their movements, as well as checking for a winner each time this algorithm is applied.

Testing

In-Development Testing

My In-Development Testing section will cover testing methods that I will use during the development of my game. It will help me maintain my code's stability and reduce errors found during post-development testing.

Test	Test Data	Test Type	Expected Output	Justification
Test for empty fields	Username = “” Email = “” Password = “” Confirm_password = “” create_account_button = True	**Erroneous**	**ERROR - No input entered in fields - please try again.**	This test is important because it will ensure that users don't gain access to the game without a valid account.
Test for valid login credentials	Username = “validuser” Password = “validpass” Login_button = True	**Normal**	**Login Successful!** **menuPage()**	This test will make sure that the login section is working correctly. To validate each credential, a database check will be performed, which will then permit a successful login and access to the main program.
Test for invalid login credentials	Username = “invaliduser” Password = “invalidpass” Login_button = True	**Erroneous**	**ERROR - Invalid credentials - please try again.**	This credential test will check to see if the details are recognised and found within the game's database, preventing anyone from entering any random credentials and gaining access to the program.
Test for a valid email format	Email = “valid@gmail.com”	**Normal**	**Email accepted.**	A valid email will need to be provided to allow a user to register successfully to their account. It will also make various accounts within my game easier to identify uniquely.
Test for an invalid email format	Email = “invalid.com”	**Erroneous**	**ERROR - Email is in the wrong format - please try again.**	
Test for long/short username (4-16 characters)	Username = “ThisUsernameIsTooLong” **OR** Username = “jade”	**Erroneous** **Boundary**	**Username accepted.** **ERROR - The username is too short - please try again.**	The username length will be a requirement in my game to allow users to have unique usernames that are different to one another. This will make separating statistics easier.

Test for long/short password (8-16 characters)	Password = Short **OR** Password = ThisPasswordIsTooLong	**Erroneous** **Erroneous**	**ERROR - The password is too short - please try again.** **ERROR - The password is too long - please try again.**	My password must be in this range as it will increase the security of a user's account. It makes brute force attacks extremely hard to perform, and would not be too hard for a user to remember.
Test for matching passwords	Password = Cinco2025 Confirm_password = Cinco2025	**Normal**	**Password accepted.**	Both passwords within the reset password section must match to ensure the right password is chosen to be linked to a user's account. Without this, the user risks losing their own account.
Test for non matching passwords	Password = Cinco2025 Confirm_password = Cinco2026	**Erroneous**	**ERROR - Passwords do not match - please try again.**	
Test for special character in password	Password = "thebest@cards"	**Normal**	**Password accepted.**	The use of a special character will be an important check within a user's password to increase its complexity. This will reduce the risk of unwanted people gaining access to a user's account.
Test for no special character in password	Password = "thebestatcards"	**Erroneous**	**ERROR - Password does not contain a special character - please try again.**	
Test for valid number of starting players (1-4)	startingPlayers = 3	**Normal**	**Integer accepted.**	The number of starting players is an integer that I will need the user to input in the game settings. As this game cannot be played with only the user, an input will be required to allow gameplay to occur.
Test for invalid number of starting players (1-4)	startingPlayers = 4	**Boundary**	**Integer accepted.**	
Test for valid number of starting cards (5-7)	startingCards = 5	**Normal**	**Integer accepted.**	The number of starting cards is a crucial integer input required for game generation. It will allow cards to be dealt to different players, allowing plays to occur. Additionally the range I have used will prevent the game from becoming too repetitive and hard to reach a resolve.
Test for invalid number of starting cards (5-7)	startingCards = 12	**Erroneous**	**ERROR - Number of starting cards out of range - please try again.**	

Test for valid difficulty (Easy, Medium, Hard)	difficulty = “Hard”	**Normal**	**Difficulty accepted.**	This test will be necessary in allowing my game to generate a skill-based game, relying on the user’s customization

Post-Development Testing

The Post-Development Testing section will involve a series of tests that will be implemented once my program is completed. These tests will confirm that every feature is working and interacting as intended. Unlike the In-Development Testing section, these checks will be performed on the finished program and will be focused on areas such as game logic, leaderboard data, the login system and menu navigation. During the testing, I will need to focus on these 3 essential elements:

Functionality Testing

Functionality testing is significant in ensuring that each feature of my game is working and interacting as intended. Examples of when I will need to use this test include within the login system, gameplay and the leaderboard. Without functionality tests, no matter the size of the feature, the user may be prevented from progressing within the game, logic errors may be accumulated, and incorrect data may be stored as a result. Strong functionality will allow the program to deliver a complete experience to a user.

Example: Login System: If a correct username and password are entered, then the user should be able to access the main menu.

Robustness Testing

Robustness testing is necessary in the program's response to invalid and unexpected data without printing errors during execution. A user may accidentally enter incorrect data, such as a wrong password or leaving inputs blank. A robust system should handle these mistakes with helpful pointers. Robustness will protect the program from crashes, improving the reliability and stability of the game.

Example: Empty username input: Instead of producing an error, the program should prompt the user to enter an actual username.

Usability Testing

Usability testing is crucial in ensuring that the program is easy, accessible, and intuitive to use. My stakeholders will play a big role in this process, as feedback from their user experiences will give me different perspectives on clarity, layout, navigation and responsiveness. Usability testing will help me provide an enjoyable and user-friendly experience that satisfies the target audience.

Example: Stakeholder plays game and wins: I will then ask the user several questions about the program as a whole, such as:

- Is the menu easy to navigate and understand?
- Were you confused at any point within my game?
- Are the error messages helpful and clear?

After my game has been fully developed, I will create a detailed Post-Development Testing table, similar to what I have done in the previous section. It will show the type of test, purpose, input, expected output, actual output and justification for a valid or invalid result. The final testing will be key in ensuring that my program is fully functional and error-free.

Development

Prototype 1

Within prototype 1 I will be focusing on creating a user-friendly GUI which will include the title page, login page, register page, reset password page and menu screen. Each page will link to one another via buttons. Additionally I will be creating a login system which will allow users to create an account, reset their password or log into my game. For this to be possible, I will also need to create a database, to store the details of existing users for login validation, as well as to add new details from newly registered users. As this is a large amount of tasks to complete, I have created a list below, which I can refer to during the development of prototype 1:

Tasks:

- Title page GUI
- Login page GUI
- Register page GUI
- Reset Password page GUI
- Menu page GUI
- Database to store and check credentials
- Working buttons

Title Page GUI

The first thing I will start off with is creating my title page GUI. As this is the very first thing that the user will see, it is important that it is executed correctly. I started by importing and initialising pygame, and creating a solid background for my game.

```
import pygame
#importing libraries

pygame.init()
#initialising pygame

title_page = pygame.display.set_mode( size: (0,0), pygame.FULLSCREEN)
title_page.fill("#cd2626")
```

This has led to my first error. Although it shows that the code has correctly been run, no screen has appeared. This is because I have forgotten to add the loop which keeps the display on-screen. I have fixed this issue on the next page:

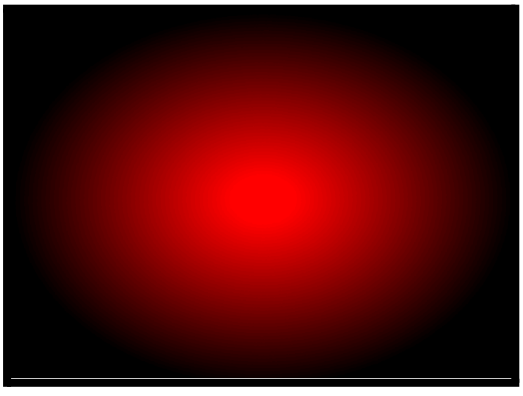
```
C:\Users\madeb\PycharmProjects\PythonProject\.venv\Scripts\python.exe "C:\Users\madeb\PycharmProjects\PythonProject\Jaden's NEA(pygame).py"
pygame 2.6.1 (SDL 2.28.4, Python 3.13.9)
Hello from the pygame community. https://www.pygame.org/contribute.html

Process finished with exit code 0
```

```python
import pygame
#importing libraries

pygame.init()
#initialising pygame

title_page = pygame.display.set_mode( size: (0,0), pygame.FULLSCREEN)
title_page.fill("#cd2626")

runtime = True
while runtime:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            runtime = False
    pygame.display.flip()

pygame.quit()
```

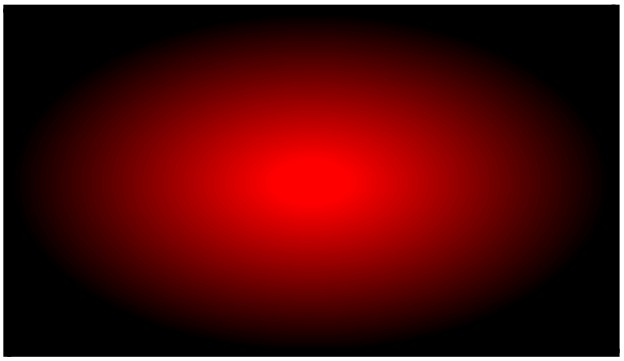
Image: Next, I decided to test whether it would be possible to add an image as a background in my game and tested if the game would accept this. This would save me time from needing to write more code to create an effect on my background. I used AI to help me not only put the picture on the screen, but also to adjust the necessary dimensions, so that the image fits to the resolution of any other screen. To ensure this fully works, I tested the code on my laptop's display (1920x1200) and an external monitor's display (1920x1080):

```python
#title page creation
title_page = pygame.display.set_mode( size: (0,0), pygame.FULLSCREEN)

#load original image
titlebg = pygame.image.load("Frontpage.png").convert()
#scale to full screen
title_w, title_h = title_page.get_size()
titlebg = pygame.transform.smoothscale(titlebg,  size: (title_w, title_h))

#gameloop
runtime = True
while runtime:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            runtime = False

    title_page.blit(titlebg,  dest: (0,0))
    pygame.display.flip()

pygame.quit()
```

Laptop display:



External monitor display:



Page Title: Now that my background is sorted, I focused on making a title for the page. As it would be repetitive to keep writing excessive lines of code just for one title, I decided to

create a class called **Title**, that I can reuse for all the pages in my game, which will display a title on the screen.

```python
#title class
class Title:  1 usage
    def __init__(self,text,font,color,x,y):
        self.text = font.render(text,True,color)
        self.rect = self.text.get_rect(center=(x,y))

    def draw(self, surface):  3 usages (2 dynamic)
        surface.blit(self.text,self.rect)

#title font
title_font = pygame.font.Font( name: "Jomhuria-Regular.ttf",  size: 300)
#title creation(main)
title_main = Title( text: "CINCO!",title_font, color: "white",title_w//2, y: 300)

#gameloop
runtime = True
while runtime:
   for event in pygame.event.get():
       if event.type == pygame.QUIT:
           runtime = False

   title_page.blit(titlebg,  dest: (0,0))
   title_main.draw(title_page)
   pygame.display.flip()

pygame.quit()
```

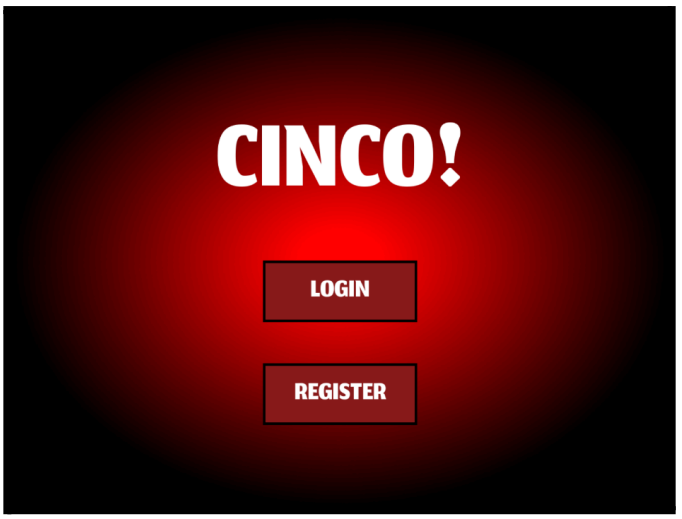
<u>Buttons:</u> Similarly to the title, I decided it would be efficient to create a class for my buttons also, as I will be reusing them many times throughout the program. This will help me to efficiently create buttons for different sections of the game.

```python
#button class
class Button():  2 usages
    def __init__(self,x,y,w,h,text,font,color,hover_color,text_color):
        self.rect = pygame.Rect(x,y,w,h)
        self.text = font.render(text,True,color)
        self.text_rect = self.text.get_rect(center=self.rect.center)
        self.color = color
        self.hover_color = hover_color
    def draw(self, surface):  3 usages (1 dynamic)
        mouse_pos = pygame.mouse.get_pos()
        if self.rect.collidepoint(mouse_pos):
            pygame.draw.rect(surface,self.hover_color,self.rect)
        else:
            pygame.draw.rect(surface,self.color,self.rect)
        surface.blit(self.text, self.text_rect)

    def is_clicked(self, event):
        if event.type == pygame.MOUSEBUTTONDOWN:
            if event.button == 1:
                if self.rect.collidepoint(event.pos):
                    return True
        return False
#button font and position
button_x = title_w // 2 - 150
button_y = title_h // 2
button_font = pygame.font.Font( name: "Jomhuria-Regular.ttf",  size: 50)

#title page buttons
login_button = Button(
    x=button_x,y=button_y,w=300,h=120,text="LOGIN",
    font=button_font,color="#8b1a1a",hover_color="#8B0000",text_color="#000000")

reg_button = Button(
    x=button_x,y=button_y + 200,w=300,h=120,text="REGISTER",
    font=button_font,color="#8b1a1a",hover_color="#8B0000",text_color="#000000")
```

This has sparked 2 issues in my code. Firstly, although I set the colour of the text in both buttons to “‘000000”, the text does not show on either button. This will set the text colour to the colour of the button instead of the chosen text colour. This is because in my definition of self.text, the text_color variable is not being used. Secondly, the bottom button, the register button, is

close to the edge of my background's gradient, and starts to blend with the background, which may act as a visual issue for my users. To fix both problems, I will include the text_color variable within the definition of self.text, and I will add a border to the buttons, by drawing an outline around the buttons. This will require new variables such as "outline_color and outline_width". Here is the fixed code below:

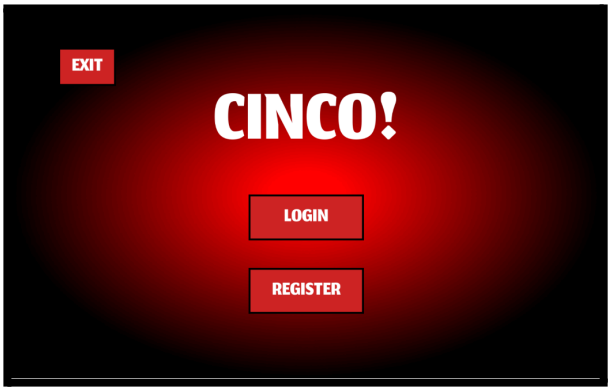
```
#button class
class Button:  2 usages
    def __init__(self,x,y,w,h,text,font,color,hover_color,text_color,border_color,border_width):
        self.rect = pygame.Rect(x,y,w,h)
        self.text = font.render(text,True,text_color)
        self.text_rect = self.text.get_rect(center=self.rect.center)
        self.color = color
        self.hover_color = hover_color
        self.border_color = border_color
        self.border_width = border_width
    def draw(self, surface):  3 usages (1 dynamic)
        mouse_pos = pygame.mouse.get_pos()
        if self.rect.collidepoint(mouse_pos):
            pygame.draw.rect(surface,self.hover_color,self.rect)
        else:
            pygame.draw.rect(surface,self.color,self.rect)
        surface.blit(self.text, self.text_rect)

        pygame.draw.rect(surface,self.border_color,self.rect,width=self.border_width)

    def is_clicked(self, event):
        if event.type == pygame.MOUSEBUTTONDOWN:
            if event.button == 1:
                if self.rect.collidepoint(event.pos):
                    return True
        return False
#button font and position
button_x = title_w // 2 - 150
button_y = title_h // 2
button_font = pygame.font.Font( name: "Jomhuria-Regular.ttf",  size: 80)

#title page buttons
login_button = Button(
    x=button_x,y=button_y,w=300,h=120,text="LOGIN",font=button_font,
    color="#8b1a1a",hover_color="#8B0000",text_color="#FFFFFF",border_color="#000000",border_width=5)

reg_button = Button(
    x=button_x,y=button_y + 200,w=300,h=120,text="REGISTER",font=button_font,
    color="#8b1a1a",hover_color="#8B0000",text_color="#FFFFFF",border_color="#000000",border_width=5)
```



I also decided to create an "EXIT" button which will allow users to exit the game from the title page. This will prevent force closing, which may cause issues to the game. I also decided to change the colour of the buttons and position buttons based on percentages instead of addition and subtraction, as they will appear similar on different screens. Finally, I adapted the beginning of my code as I realised that the program was not filling my laptop screen correctly.

```
#find pygame resolution
info = pygame.display.Info()
title_w, title_h = info.current_w, info.current_h

#title page creation
title_page = pygame.display.set_mode( size: (title_w,title_h), pygame.FULLSCREEN | pygame.SCALED)
```

```
#title page buttons
login_button = Button(
    x=button_x,y=button_y,w=300,h=120,text="LOGIN",font=button_font,
    color="#CD2626",hover_color="#8B0000",text_color="#FFFFFF",border_color="#000000",border_width=5)

reg_button = Button(
    x=button_x,y=button_y * 1.4,w=300,h=120,text="REGISTER",font=button_font,
    color="#CD2626",hover_color="#8B0000",text_color="#FFFFFF",border_color="#000000",border_width=5)

exit_button = Button(
    x=button_x * 0.2,y=button_y * 0.2,w=150,h=100,text="EXIT",font=button_font,
    color="#CD2626",hover_color="#8B0000",text_color="#FFFFFF",border_color="#000000",border_width=5)
```



Login Page GUI

Next, I decided to move on to the login page. At the moment, in this section, I will only focus on the GUI before I work on input from the user, and making sure the buttons switch between pages. Firstly, I created the new page, using the same image as the background, and added the “LOGIN” title to the page.


```
#drawing login page screen/labels/buttons
login_page.blit(loginbg,  dest: (0,0))
login_title.draw(login_page)
```

```
#login page creation
login_page = pygame.display.set_mode( size: (title_w,title_h), pygame.FULLSCREEN | pygame.SCALED)
loginbg = pygame.transform.smoothscale(titlebg,  size: (title_w, title_h))

#login page title
login_title = Title( text: "LOGIN",title_font, color: "white",title_w//2,title_h * 0.3)
```

Labels and input: In this stage, I will be creating 2 areas for users to input data: for entering a username, and for entering a password. I will have the corresponding label for each box to the left, similar to the GUI, and there will be a button at the bottom of the page, allowing users to login when my database has been setup. To create the labels, I will use my title class, and to create the buttons, I will use my button class. Additionally, I have decided to create an **InputBox** class, which I will be able to use in other areas of the game which require user input, such as registration, reset password, and game settings.

```
#class for input boxes
class InputBox:  2 usages
    def __init__(self,x,y,w,h,text_colour,font):
        self.font = font
        self.inputBox = pygame.Rect(x,y,w,h)
        self.text = " "
        self.text_colour = text_colour
        self.colourInactive = "#FFFFFF"
        self.colourActive = "#CCCCCC"
        self.colour = self.colourInactive
        self.hover_colour = "#5F5F5F"
        self.border_colour = "#000000"
        self.border_width = 5
        self.active = False
        self.inactive = True

    def do_event(self,event):  2 usages
        if event.type == pygame.MOUSEBUTTONDOWN:
            self.active = self.inputBox.collidepoint(event.pos)
            if self.active:
                self.colour = self.colourActive
            else:
                self.colour = self.colourInactive
        if event.type == pygame.KEYDOWN:
            if self.active:
                if event.key == pygame.K_RETURN:
                    print(self.text)
                elif event.key == pygame.K_BACKSPACE:
                    self.text = self.text[:-1]
                else:
                    self.text += event.unicode
```

```
login_username_input.draw(login_page)
login_password_input.draw(login_page)
```

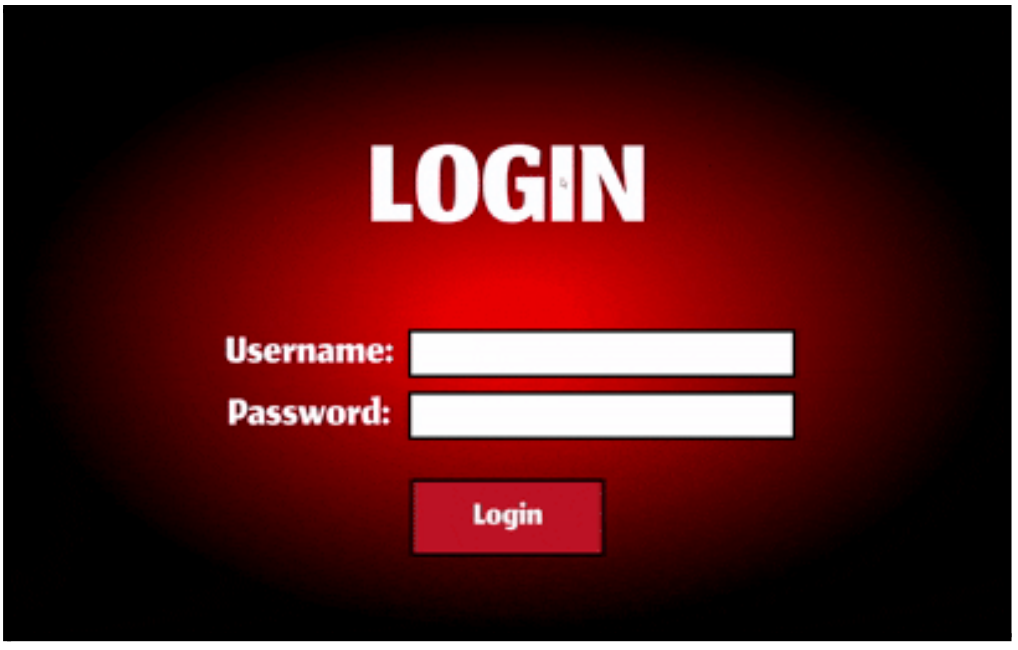
```
    def draw(self, surface):  3 usages (1 dynamic)
        textsurface = self.font.render(self.text, True, self.colour)
        width = max(200,textsurface.get_width()+10)
        self.inputBox.w = width
        surface.blit(textsurface,(self.inputBox.x+5,self.inputBox.y+5))
        pygame.draw.rect(surface,self.colour,self.inputBox)
        mouse_pos = pygame.mouse.get_pos()
        if self.inputBox.collidepoint(mouse_pos):
            pygame.draw.rect(surface, self.hover_colour, self.inputBox)
        else:
            pygame.draw.rect(surface, self.colour, self.inputBox)

#login input boxes
login_username_input = InputBox(title_w * 0.4,title_h * 0.55, w: 300, h: 120, text_colour: "#FFFFFF",normal_font)
login_password_input = InputBox(title_w * 0.4,title_h * 0.65, w: 300, h: 120, text_colour: "#FFFFFF",normal_font)
```

As shown by the GIF, the input boxes are working fine. They take entered letters from my keyboard and I can use my backspace buttons to delete the letters I have entered. However, I have positioned these boxes in the wrong positions, and the normal_font variable is set to have white text instead of black text, and as the colour of the box when it is hovered over is almost identical to the font colour, users will struggle to see what they are typing. Additionally, I have not drawn on the border correctly in my code, as it does not show up in the main program. To fix this, I will make sure that the border of each box, and the text of the user input is displayed correctly. I will also make sure that the boxes are positioned and sized so they lie next to their respective labels. Additionally, I made the hover colour and active colour of each input box easier to differentiate so that the user will know what box they are typing within. The final result is shown below:

```
def draw(self, surface):  3 usages (1 dynamic)
    mouse_pos = pygame.mouse.get_pos()
    if self.inputBox.collidepoint(mouse_pos):
        pygame.draw.rect(surface, self.hover_colour, self.inputBox)
    else:
        pygame.draw.rect(surface, self.colour, self.inputBox)
    textsurface = self.font.render(self.text, True, self.text_colour)
    width = max(600,textsurface.get_width()+10)
    self.inputBox.w = width
    surface.blit(textsurface,(self.inputBox.x+5,self.inputBox.y+5))
    pygame.draw.rect(surface, self.border_colour, self.inputBox, width=self.border_width)
```

```python
#class for input boxes
class InputBox:  2 usages
    def __init__(self,x,y,w,h,text_colour,font):
        self.rect = pygame.Rect(x,y,w,h)
        self.font = font
        self.text_colour = text_colour
        self.inputBox = pygame.Rect(x,y,w,h)
        self.text = " "
        self.colourInactive = "#FFFFFF"
        self.colourActive = "#808080"
        self.colour = self.colourInactive
        self.hover_colour = "#5F5F5F"
        self.border_colour = "#000000"
```

```python
#login input boxes
login_username_input = InputBox(title_w * 0.4,title_h * 0.51, w: 300, h: 75, text_colour: "#000000",normal_font)
login_password_input = InputBox(title_w * 0.4,title_h * 0.61, w: 300, h: 75, text_colour: "#000000",normal_font)

#login button
login_page_button = Button(x=button_x,y=button_y * 1.5,w=300,h=120,text="Login",font=button_font,
    colour="#CD2626",hover_colour="#8B0000",text_colour="#FFFFFF",border_colour="#000000",border_width=5)
```



Register Page GUI

Next, i moved on to the register page. This will be the main page where new users will need to create an account in order to progress to the main game. It is important that this page functions correctly.

Background: Just like the Login and Title Pages, I started off by building the code for the background which will include the title and the image i will use. I did this by using similar code from the last 2 GUIs and using my **Title** class to create a title for the page:

```
#register page creation
register_page = pygame.display.set_mode( size: (title_w,title_h), pygame.FULLSCREEN | pygame.SCALED)
registerbg = pygame.transform.smoothscale(titlebg,  size: (title_w,title_h))

#register page title
register_title = Title( text: "REGISTER",title_font, colour: "white",title_w//2,title_h * 0.3)
```

```
#drawing register page screen/labels/buttons
register_page.blit(registerbg,  dest: (0,0))
register_title.draw(register_page)
```

Labels and input: Just like the Login page, the Register page will require inputs from the user in order to progress. I will create text boxes and labels which will be associated with 4 details: username, email address, password and the confirmed password. Again, I will use the **Title** class and newly created **InputBox** class.

```
#register page labels and inputs
reg_username_label = Title( text: "Username:",normal_font, colour: "white",title_w * 0.3,title_h * 0.45)
reg_username_input = InputBox(title_w * 0.4,title_h * 0.41, w: 300, h: 75, text_colour: "#000000",normal_font)
reg_email_label = Title( text: "Email Address",normal_font, colour: "white",title_w * 0.3,title_h * 0.55)
reg_email_input = InputBox(title_w * 0.4,title_h * 0.51, w: 300, h: 75, text_colour: "#000000",normal_font)
reg_password_label = Title( text: "Password:",normal_font, colour: "white",title_w * 0.3,title_h * 0.65)
reg_password_input = InputBox(title_w * 0.4,title_h * 0.61, w: 300, h: 75, text_colour: "#000000",normal_font)
reg_confirm_label = Title( text: "Confirm Password:",normal_font, colour: "white",title_w * 0.3,title_h * 0.75)
reg_confirm_input = InputBox(title_w * 0.4,title_h * 0.71, w: 300, h: 75, text_colour: "#000000",normal_font)
```

```
#drawing register page screen/labels/buttons
register_page.blit(registerbg, dest: (0,0))
register_title.draw(register_page)
reg_username_label.draw(register_page)
reg_username_input.draw(register_page)
reg_email_label.draw(register_page)
reg_email_input.draw(register_page)
reg_password_label.draw(register_page)
reg_password_input.draw(register_page)
reg_confirm_label.draw(register_page)
reg_confirm_input.draw(register_page)
pygame.display.flip()
```

```
#register input text
reg_username_input.do_event(event)
reg_email_input.do_event(event)
reg_password_input.do_event(event)
reg_confirm_input.do_event(event)
```

Although the code is running correctly, the labels and the input boxes are positiioned incorrectly. I will need to move the labels back so that it is clear for the user to see which input box is assigned to each label, avoiding confusion. I have also adjusted the position of the “create account” button to center it, as the previous parameter “buttoon_x” only applies a button to the centre when the button has a width of 300, which has changed to 400 ,which was done to display the full text. This has all been fixed below:

```
#register page labels and inputs
reg_username_label = Title( text: "Username:",normal_font, colour: "white",title_w * 0.3,title_h * 0.45)
reg_username_input = InputBox(title_w * 0.4,title_h * 0.41, w: 300, h: 75, text_colour: "#000000",normal_font)
reg_email_label = Title( text: "Email Address:",normal_font, colour: "white",title_w * 0.27,title_h * 0.55)
reg_email_input = InputBox(title_w * 0.4,title_h * 0.51, w: 300, h: 75, text_colour: "#000000",normal_font)
reg_password_label = Title( text: "Password:",normal_font, colour: "white",title_w * 0.3,title_h * 0.65)
reg_password_input = InputBox(title_w * 0.4,title_h * 0.61, w: 300, h: 75, text_colour: "#000000",normal_font)
reg_confirm_label = Title( text: "Confirm Password:",normal_font, colour: "white",title_w * 0.24,title_h * 0.75)
reg_confirm_input = InputBox(title_w * 0.4,title_h * 0.71, w: 300, h: 75, text_colour: "#000000",normal_font)

#create account button
create_account_button = Button(x=(title_w - 400) // 2,y=button_y * 1.65,w=400,h=120,text="Create Account",font=button_font,
    colour="#CD2626",hover_colour="#8B0000",text_colour="#FFFFFF",border_colour="#000000",border_width=5)
```

```
#drawing register page screen/labels/buttons
register_page.blit(registerbg, dest: (0,0))
register_title.draw(register_page)
reg_username_label.draw(register_page)
reg_username_input.draw(register_page)
reg_email_label.draw(register_page)
reg_email_input.draw(register_page)
reg_password_label.draw(register_page)
reg_password_input.draw(register_page)
reg_confirm_label.draw(register_page)
reg_confirm_input.draw(register_page)
create_account_button.draw(register_page)
```

Reset Password Page GUI

For the next GUI, I will be creating the reset password page, where users will be directed to reset their password if it has been forgotten. However, upon reflection, I have realised that I have not included this button in the Login Page GUI. I first started by creating this button.

```
#login page buttons
login_page_button = Button(x=button_x * 0.6,y=button_y * 1.5,w=300,h=120,text="Login",font=button_font,
    colour="#CD2626",hover_colour="#8B0000",text_colour="#FFFFFF",border_colour="#000000",border_width=5)
forgot_password_button = Button(x=button_x * 1.3,y=button_y * 1.5,w=400,h=120,text="Forgot Password",font=button_font,
    colour="#CD2626",hover_colour="#8B0000",text_colour="#FFFFFF",border_colour="#000000",border_width=5)
```

```
login_page_button.draw(login_page)
forgot_password_button.draw(login_page)
```

Background: Next, I moved on to writing the code to display the background, which I will keep the same as the previous pages.

```
#reset page creation
reset_page = pygame.display.set_mode( size: (title_w,title_h), pygame.FULLSCREEN | pygame.SCALED)
resetbg = pygame.transform.smoothscale(titlebg,  size: (title_w,title_h))

#reset page title
reset_title = Title( text: "RESET PASSWORD",title_font, colour: "white",title_w//2,title_h * 0.3)
```

```
#drawing reset password page screen/labels/buttons
reset_page.blit(resetbg, dest: (0,0))
reset_title.draw(reset_page)
```

Labels and input: Now I will add the labels,"Email Address", "Password" and "Confirm Password", along with input boxes for each, to let the user know which box is for which input. I have also adjusted the x position of all the labels to ensure they line up with their designated box, similar to the register page.

```
#reset page labels and inputs
reset_email_label = Title( text: "Email Address:",normal_font, colour: "white",title_w * 0.27,title_h * 0.45)
reset_email_input = InputBox(title_w * 0.4,title_h * 0.41, w: 300, h: 75, text_colour: "#000000",normal_font)
reset_password_label = Title( text: "Password:",normal_font, colour: "white",title_w * 0.3,title_h * 0.55)
reset_password_input = InputBox(title_w * 0.4,title_h * 0.51, w: 300, h: 75, text_colour: "#000000",normal_font)
reset_confirm_label = Title( text: "Confirm Password:",normal_font, colour: "white",title_w * 0.24,title_h * 0.65)
reset_confirm_input = InputBox(title_w * 0.4,title_h * 0.61, w: 300, h: 75, text_colour: "#000000",normal_font)
```

```
#drawing reset password page screen/labels/buttons
reset_page.blit(resetbg, dest: (0,0))
reset_title.draw(reset_page)
reset_email_label.draw(reset_page)
reset_email_input.draw(reset_page)
reset_password_label.draw(reset_page)
reset_password_input.draw(reset_page)
reset_confirm_label.draw(reset_page)
reset_confirm_input.draw(reset_page)
```

As shown by the video, the labels and input boxes are all in the correct positions. Unfortunately, I am unable to type in the boxes because I have not included the function which detects keys entered within the game loop. While also fixing this issue in the code

below, I also instantiated a new button which users can click on once their credentials have been inputted. Additionally, i have renamed “Password” to “New Password” to signify that the user must enter a new password. The final code and output are shown below:

```
#reset page labels and inputs
reset_email_label = Title( text: "Email Address:",normal_font, colour: "white",title_w * 0.27,title_h * 0.45)
reset_email_input = InputBox(title_w * 0.4,title_h * 0.41, w: 300, h: 75, text_colour: "#000000",input_font)
reset_password_label = Title( text: "Password:",normal_font, colour: "white",title_w * 0.3,title_h * 0.55)
reset_password_input = InputBox(title_w * 0.4,title_h * 0.51, w: 300, h: 75, text_colour: "#000000",input_font)
reset_confirm_label = Title( text: "Confirm Password:",normal_font, colour: "white",title_w * 0.24,title_h * 0.65)
reset_confirm_input = InputBox(title_w * 0.4,title_h * 0.61, w: 300, h: 75, text_colour: "#000000",input_font)

#reset password button
reset_button = Button(x=(title_w - 400) // 2,y=button_y * 1.5,w=400,h=120,text="Reset Password",font=button_font,
    colour="#CD2626",hover_colour="#8B0000",text_colour="#FFFFFF",border_colour="#000000",border_width=5)
```

```
#reset input text
reset_email_input.do_event(event)
reset_password_input.do_event(event)
reset_confirm_input.do_event(event)
```

```
reset_button.draw(reset_page)
```

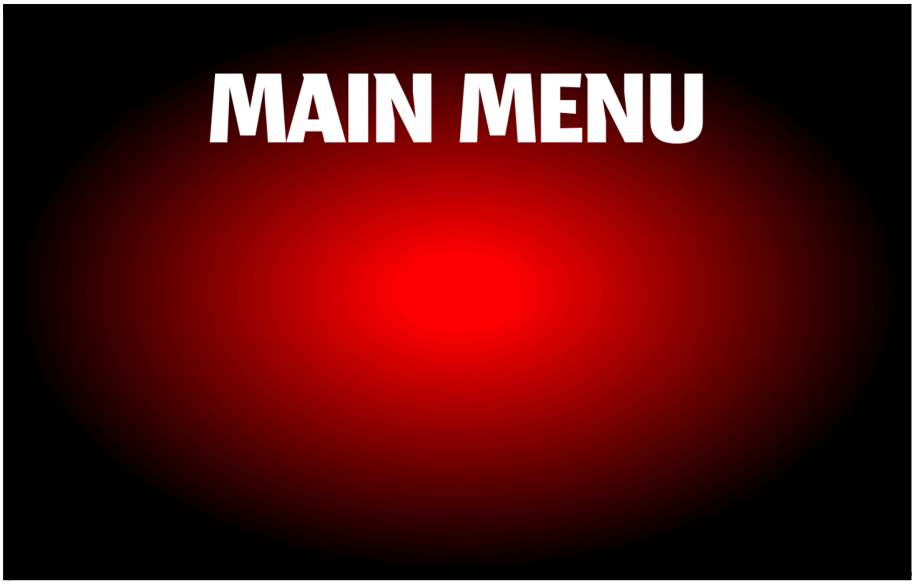
Menu Page GUI

Now I will move on to creating the page which will allow users to choose the page they would like to access. However I will only use this section to create the main menu page, and I will save creating individual pages such as the leaderboard, for **Prototype 2**.

Background: Just like before, I will use the same code to create the background and title for my page, with only the string used for the title changing to “MAIN MENU”.

```
#drawing menu page screen/labels/buttons
menu_page.blit(menubg, dest: (0,0))
menu_title.draw(menu_page)
```

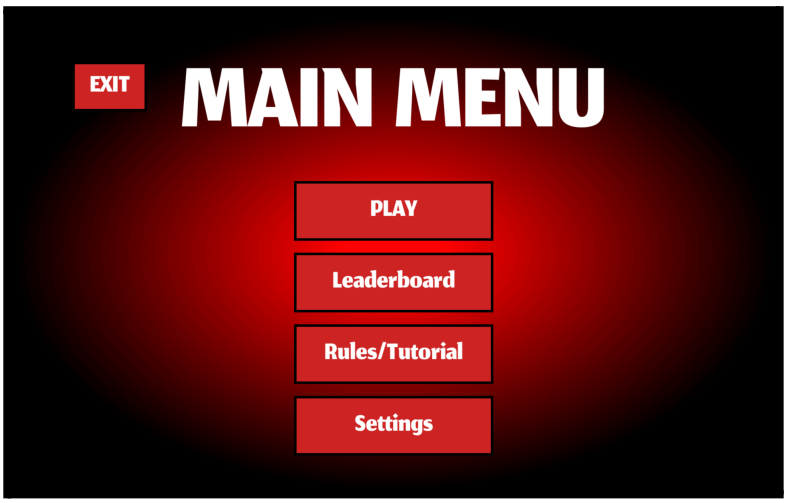
```
#menu page creation
menu_page = pygame.display.set_mode( size: (title_w,title_h), pygame.FULLSCREEN | pygame.SCALED)
menubg = pygame.transform.smoothscale(titlebg,  size: (title_w,title_h))

#menu page title
menu_title = Title( text: "MAIN MENU",title_font, colour: "white",title_w//2,title_h * 0.2)
```



Buttons: Now I will need to add the buttons to each page, as they will be necessary in switching between pages. I have included buttons for actual gameplay, the leaderboard, the rules and the settings of the game, as well as an exit button for users to leave the game. Here is the code and output below, working as intended:

```
#menu page buttons
menu_play_button = Button(x=(title_w - 400) // 2,y=button_y * 0.7,w=400,h=120,text="PLAY",font=button_font,
    colour="#CD2626",hover_colour="#8B0000",text_colour="#FFFFFF",border_colour="#000000",border_width=5)

menu_leaderboard_button = Button(x=(title_w - 400) // 2,y=button_y * 1,w=400,h=120,text="Leaderboard",font=button_font,
    colour="#CD2626",hover_colour="#8B0000",text_colour="#FFFFFF",border_colour="#000000",border_width=5)

menu_rules_button = Button(x=(title_w - 400) // 2,y=button_y * 1.3,w=400,h=120,text="Rules/Tutorial",font=button_font,
    colour="#CD2626",hover_colour="#8B0000",text_colour="#FFFFFF",border_colour="#000000",border_width=5)

menu_settings_button = Button(x=(title_w - 400) // 2,y=button_y * 1.6,w=400,h=120,text="Settings",font=button_font,
    colour="#CD2626",hover_colour="#8B0000",text_colour="#FFFFFF",border_colour="#000000",border_width=5)

menu_exit_button = Button(x=button_x * 0.2,y=button_y * 0.2,w=150,h=100,text="EXIT",font=button_font,
    colour="#CD2626",hover_colour="#8B0000",text_colour="#FFFFFF",border_colour="#000000",border_width=5)
```

```
#drawing menu page screen/labels/buttons
menu_page.blit(menubg, dest: (0,0))
menu_title.draw(menu_page)
menu_play_button.draw(menu_page)
menu_leaderboard_button.draw(menu_page)
menu_rules_button.draw(menu_page)
menu_settings_button.draw(menu_page)
menu_exit_button.draw(menu_page)
```



Game Database

Next, I moved on to creating my game's database. This will be necessary, especially for the login system, as it will allow the game to quickly filter whether given credentials are valid or not. To effectively tackle this problem, I will split it into 2 sections; the **creation of the database** and **testing of data**.

Database Creation

For this project I have decided to use **sqlite3** for my game, as it provides efficient data access and reliability, and should be able to store necessary data from my game. Before any coding can begin, I started by importing sqlite3 to the program:

```
#importing libraries
import pygame
import sqlite3
```

```
#creating database

#connecting to database file
credentials_db = sqlite3.connect("credentials.db")

#creating cursor, to send commands to the database
cursor = credentials_db.cursor()
```

Database file: Next, I created the file that will hold the credentials of the users. I did this using the .conn command, which is used for connecting to the database, but can also be used to create a database if it does not exist already. I also added a cursor to allow commands to be executed relating to the database.

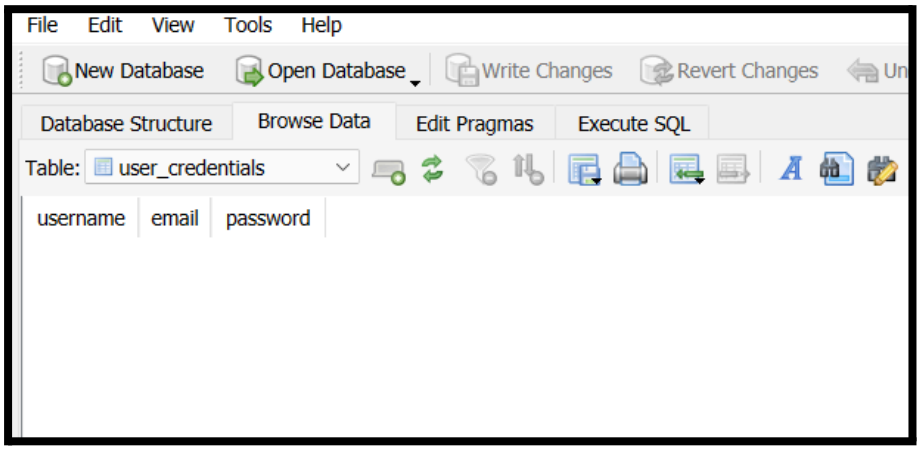
Table Of Data: Now that a file has been created, I can import a table which will store all of the values. For now, this will hold the username, email address and password of each user.

```
#creating table of data
cursor.execute("""
    CREATE TABLE IF NOT EXISTS user_credentials (
        username TEXT NOT NULL,
        email TEXT NOT NULL,
        password TEXT NOT NULL,
    )""")
```

```
    cursor.execute("""
    ~~~~~~~~~~~~~~~^^^^
        CREATE TABLE IF NOT EXISTS users (
        ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
    ...<4 lines>...
            )
            ^
    """)
    ^^^^
sqlite3.OperationalError: near ")": syntax error
```

Unfortunately, an error has occurred, and I realised that I included an unnecessary comma after the line with password. I have fixed this code below, and opened my database in DB browser for SQLite to ensure the table has been created correctly:

```
#creating table of data
cursor.execute("""
    CREATE TABLE IF NOT EXISTS user_credentials (
        username TEXT NOT NULL,
        email TEXT NOT NULL,
        password TEXT NOT NULL
    )""")
```

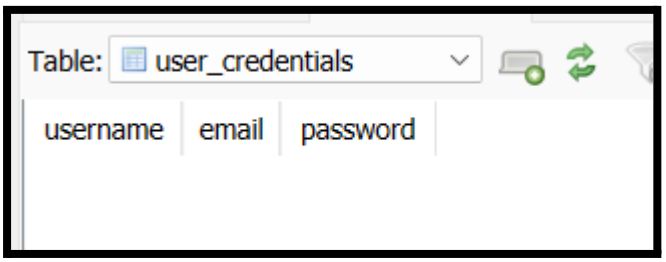


Add users: Next, I decided to create a function called addUser, which will add users to the database after an account has been registered to the game.

```
#adding users to the database
def addUser(username,email,password):
    connect = sqlite3.connect("credentials.db")
    cur = credentials_db.cursor()
    cur.execute( sql: "INSERT INTO user_credentials VALUES (?,?,?)", parameters: (username,email,password))
    connect.commit()
    connect.close()
```

To test this works, I added sample data, which I will check has been added to the database on DB Browser.

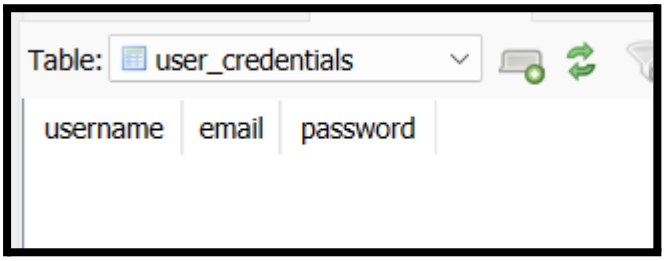
```
addUser(JadenHamilton,me@gmail.com,CINCO2025)
```



Unfortunately, the data has not appeared in the database file. Upon further reflection and revisiting the structure of the code. I have noticed 3 major errors. In my sample data, I have not included quotation marks, so the program thinks they are variables instead of strings. Also, my cursor line of code is using the wrong variable. It should say connect.cursor(), instead of credentials_db.cursor(). Finally, I did not specify what column each piece of data is to go into.

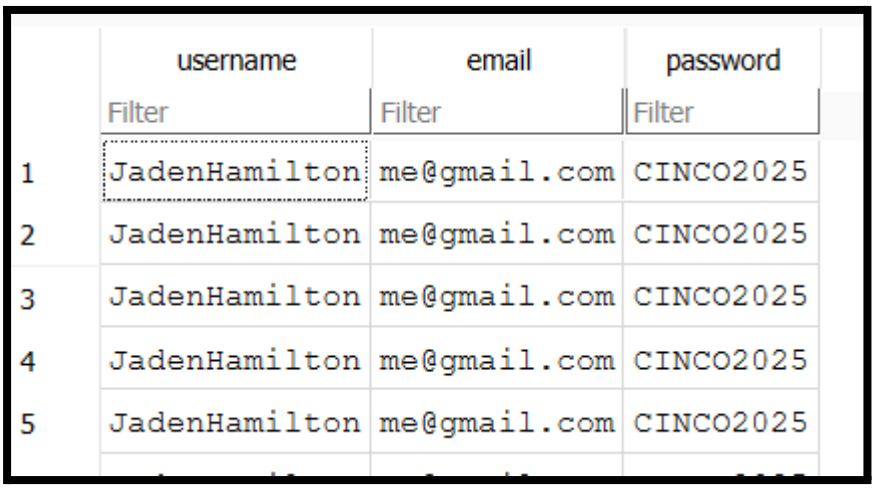
```
#adding users to the database
def addUser(username,email,password):  1 usage
    connect = sqlite3.connect("credentials.db")
    cur = connect.cursor()
    cur.execute( sql: "INSERT INTO user_credentials (username, email, password) VALUES (?,?,?)",
     parameters: (username,email,password))
    connect.commit()
    connect.close()
```

```
addUser( username: "JadenHamilton", email: "me@gmail.com", password: "CINCO2025")
```



Again, the code still does not appear within the database. However, this is the case because the function has been written outside the gameloop, so it will never be executed while the game is being run. To fix this I will move the sample data into the gameloop.

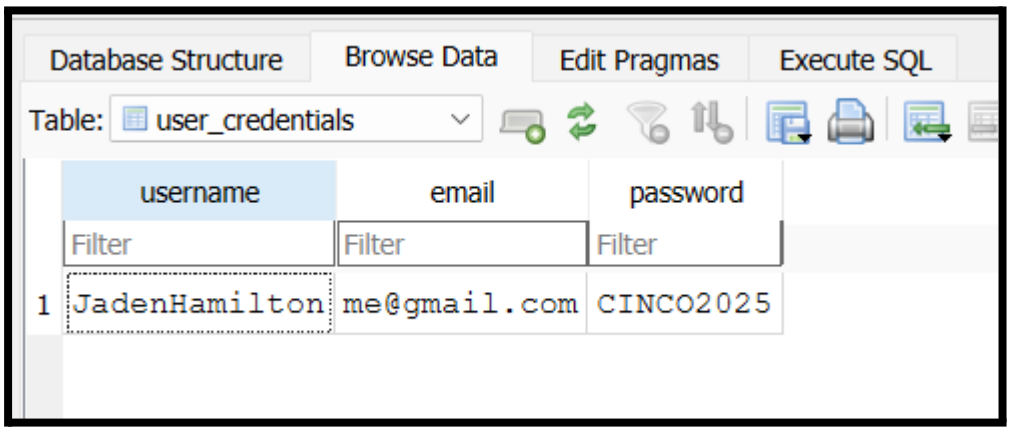
```
pygame.display.flip()
addUser( username: "JadenHamilton", email: "me@gmail.com", password: "CINCO2025")
```



	username	email	password
	Filter	Filter	Filter
1	JadenHamilton	me@gmail.com	CINCO2025
2	JadenHamilton	me@gmail.com	CINCO2025
3	JadenHamilton	me@gmail.com	CINCO2025
4	JadenHamilton	me@gmail.com	CINCO2025
5	JadenHamilton	me@gmail.com	CINCO2025

The code has worked, however because the function is in a gameloop it will never stop. This has led to repeats of the same data being added to the database. When trying to delete all of the previous entries even after closing the program, my laptop kept not responding. To safely resolve this problem, I deleted the original database, and moved the sample data just above the game loop so it doesnt repeat. As the previous code will create a database if it doesn't exist, the code will create a new program for me automatically. Here is the final code below:

```
addUser( username: "JadenHamilton", email: "me@gmail.com", password: "CINCO2025")

#gameloop
runtime = True
```



Data Testing

Now I will move on to testing data that is inputted by the user, primarily from the login, register and reset password pages. For each page I will need to perform the following validations:

- Presence Check (For ALL)
- Length Check (For Username, Password and Confirm Password)
- Format Check (For Password, Confirm Password and Email Address)
- Lookup Check (For Username and Password)
- Match Check (For Password and Confirm Password)

Login Page

Presence Check - To execute each validation, I will be creating a function for each check, starting with the presence check. I will do this using the text.strip() procedure, which removes the blank spaces from a string. By doing this, I will be able to see if a user has inputted text or just blank spaces. I have also added a statement for the program to print if a blank field has been entered. I tested this on the login page:

```
#data checks for validation

#presence check function
def presence_check(text):  2 usages
    return text.strip()

presence_check_text = Title( text: "Not all fields are filled in - Please try again.",normal_font, colour: "white",title_w//2, title_h * 0.2)
```

```
#data validation
if login_page_button.is_clicked(event):
    username = login_username_input.text.strip()
    password = login_password_input.text.strip()

    #presence check
    if not presence_check(username) or not presence_check(password):
        presence_check_text.draw(login_page)
```

Unfortunately, although I left the password entry empty, the presence check error did not appear on the screen. This is because the message only appears on the screen for a single frame due to its position in the event loop. To handle this logic error, I decided to create a variable which tracks when to display the error message, and does this in the same block of code where I display the login screen. I tried this below and here is the output:

```
#presence check
if not presence_check(username) or not presence_check(password):
    show_presence_check_error = True
```

```
#login validation variables
if show_presence_check_error:
    presence_check_text.draw(login_page)
```

```
Traceback (most recent call last):
  File "C:\Users\madeb\PycharmProjects\PythonProject\Jaden's NEA(pygame).py", line 297, in <module>
    if show_presence_check_error:
       ^^^^^^^^^^^^^^^^^^^^^^^^^
NameError: name 'show_presence_check_error' is not defined
```

I have received another error message from the program, because I have not defined my variables. To fix this, I created a designated section in my code to define variables I will use within my validation section. I also adjusted the y position of the error message so that it appears under the buttons instead of above the title.

```
#creating validation variables
show_presence_check_error = False
```

```
#presence check function
def presence_check(text):  2 usages
    return text.strip()

presence_check_text = Title( text: "Not all fields are filled in - Please try again.",normal_font, colour: "white",title_w//2, title_h * 0.95)
```

As shown in the GIF, the code is functioning correctly. However I intended on the code removing the presence check error text after all fields are filled in, which it does not after I fill in the password field. I reviewed the code again, and realised that there is no statement where the variable show_presence_check_error becomes false. This means that the error text will continue to be displayed, which is not what I want. Instead, I wrote an else statement in the selection to set show_presence_check_error to false. This will have to change in my next validation because I will need to include elif statements similar to my algorithm. Here is the final code below:

```
#presence check
if not presence_check(username) or not presence_check(password):
    show_presence_check_error = True
else:
    show_presence_check_error = False
```

```
#length check for username
elif length_check_user(username) == 1:
    show_presence_check_error = False
    show_length_check_error_user_short = True
    show_length_check_error_user_long = False
    show_length_check_error_pass_short = False
    show_length_check_error_pass_long = False

elif length_check_user(username) == 2:
    show_presence_check_error = False
    show_length_check_error_user_short = False
    show_length_check_error_user_long = True
    show_length_check_error_pass_short = False
    show_length_check_error_pass_long = False

#length check for password
elif length_check_pass(password) == 1:
    show_presence_check_error = False
    show_length_check_error_user_short = False
    show_length_check_error_user_long = False
    show_length_check_error_pass_short = True
    show_length_check_error_pass_long = False

elif length_check_pass(password) == 2:
    show_presence_check_error = False
    show_length_check_error_user_short = False
    show_length_check_error_user_long = False
    show_length_check_error_pass_short = False
    show_length_check_error_pass_long = True
```

Length Check - Next I moved on to performing length checks on my inputs. I have decided that usernames will need to be between 4-16 characters, and the passwords will need to be between 8-16 characters. Unlike the presence check, I will need to create two functions, for the username and the password. When I move onto the registration and reset password pages, I will need to create this function for the confirm password section. I will also need to create separate variables that will determine if a username is too long or short, and if a password is to long or short. As the username can only be 4-16 characters and the password can only be 8-16 characters in my game, I will add these as conditions in their respective functions. Additionally, instead of returning false and true I will return 1 and 2. Here is the code I have written for this section and the output.

```
#length check function for login system username
def length_check_user(text):  2 usages
    if len(text) < 4:
        return 1
    elif len(text) > 16:
        return 2
    return None

#length check function for login system password:
def length_check_pass(text):  2 usages
    if len(text) < 8:
        return 1
    elif len(text) > 16:
        return 2
    return None

length_check_text_user_short  = Title( text: "Username is too short - Please try again.",normal_font, colour: "white",title_w//2, title_h * 0.95)
length_check_text_user_long  = Title( text: "Username is too long - Please try again.",normal_font, colour: "white",title_w//2, title_h * 0.95)
length_check_text_pass_short = Title( text: "Password is too short - Please try again.",normal_font, colour: "white",title_w//2, title_h * 0.95)
length_check_text_pass_long  = Title( text: "Password is too long - Please try again.",normal_font, colour: "white",title_w//2, title_h * 0.95)

#creating validation variables
show_presence_check_error = False
show_length_check_error_user_short = False
show_length_check_error_user_long = False
show_length_check_error_pass_short = False
show_length_check_error_pass_long = False
```

Format Check: I will be using a format check within the login section, to ensure that a special character has been inputted into the code. To do this effectively, I imported the library “string”, and used the function string.punctuation in a list, to list all of the possible special characters, for effective filtering of characters. I also created a format check variable to display the text to notify a user to include a special character.

```
#importing libraries
import pygame
import string
import sqlite3
```

```
show_format_check_error_pass = False
```

```
#format check function for login system password
def has_special_char(text):  1 usage
    special_characters = string.punctuation
    for char in text:
        if char in special_characters:
            return True
        else:
            return False
    return None
format_check_text = Title( text: "Password must contain a special character - Please Try again.",normal_font, colour: "white",title_w//2, title_h * 0.95)
```

```
#format check for password
elif has_special_char(password):
    show_presence_check_error = False
    show_length_check_error_user_short = False
    show_length_check_error_user_long = False
    show_length_check_error_pass_short = False
    show_length_check_error_pass_long = False
    show_format_check_error_pass = True
```

Unfortunately this did not work as intended because my elif statement currently checks that there is a special character. However, I want the error message to display when there isn't a character in the password. To fix, I changed the statement to check that there isnt a special character in the password.

```
#format check for password
elif not has_special_char(password):
    show_presence_check_error = False
    show_length_check_error_user_short = False
    show_length_check_error_user_long = False
    show_length_check_error_pass_short = False
    show_length_check_error_pass_long = False
    show_format_check_error_pass = True

else:
    show_format_check_error_pass = False
```

Lookup Check: This check will ensure that the username and password entered match a username and password assigned in the database. To test this I will use the same details I previously used as sample data. I will also add a temporary label and temporary variable to signify if the login is successful, and display this output.

	username	email	password
	Filter	Filter	Filter
1	JadenHamilton	me@gmail.com	CINCO@2025

```
elif show_lookup_error:
    lookup_check_text.draw(login_page)
elif show_login_successful:
    login_successful_text.draw(login_page)
```

```
elif check_User(username,password):
    show_presence_check_error = False
    show_length_check_error_user_short = False
    show_length_check_error_user_long = False
    show_length_check_error_pass_short = False
    show_length_check_error_pass_long = False
    show_format_check_error_pass = True
    show_lookup_error = False
    show_login_successful = True
```

However, this has led to another error. The program keeps displaying the message which notifies the user that the password must contain a special character, even though mine contains the “@” symbol. I reviewed my code again, and realise that the variable to display this message was set to true when the program checks if the data is in the database. I can fix this by setting it to false, so that the program will display login successful. I have fixed this below:

```
#lookup check
elif not check_User(username,password):
    show_presence_check_error = False
    show_length_check_error_user_short = False
    show_length_check_error_user_long = False
    show_length_check_error_pass_short = False
    show_length_check_error_pass_long = False
    show_format_check_error_pass = False
    show_lookup_error = True

#lookup check for success
elif check_User(username,password):
    show_presence_check_error = False
    show_length_check_error_user_short = False
    show_length_check_error_user_long = False
    show_length_check_error_pass_short = False
    show_length_check_error_pass_long = False
    show_format_check_error_pass = False
    show_lookup_error = False
    show_login_successful = True
```

Register Page

Before starting the validation process for the registration page, I realised that I have made a vital mistake. My length and format checks should only be performed in the registration page, not the login page. This is because it will be stored in the database already, so there is no need to check this again, except for lookup and presence checks in the login page. To fix this, I kept each validation the same, but just moved each if statement to when the register button is pressed.

```
#login input text
login_username_input.do_event(event)
login_password_input.do_event(event)

#data validation for login
if login_page_button.is_clicked(event):
    username = login_username_input.text.strip()
    password = login_password_input.text.strip()

    #presence check
    if not presence_check(username) or not presence_check(password):
        show_presence_check_error = True
        show_lookup_error = False

    #lookup check
    elif not check_User(username,password):
        show_presence_check_error = False
        show_lookup_error = True

    #lookup check for success
    elif check_User(username,password):
        show_presence_check_error = False
        show_lookup_error = False
        show_login_successful = True
```

```
# register page validations
elif reg_button.is_clicked(event):
    username = reg_username_input.text.strip()
    email = reg_email_input.text.strip()
    password = reg_password_input.text.strip()
    conpassword = reg_confirm_input.text.strip()

    # length check for username
elif length_check_user(username) == 1:
    show_presence_check_error = False
    show_length_check_error_user_short = True
    show_length_check_error_user_long = False
    show_length_check_error_pass_short = False
    show_length_check_error_pass_long = False
    show_format_check_error_pass = False
    show_lookup_error = False

elif length_check_user(username) == 2:
    show_presence_check_error = False
    show_length_check_error_user_short = False
    show_length_check_error_user_long = True
    show_length_check_error_pass_short = False
    show_length_check_error_pass_long = False
    show_format_check_error_pass = False
    show_lookup_error = False
```

```
# length check for password
elif length_check_pass(password) == 1:
    show_presence_check_error = False
    show_length_check_error_user_short = False
    show_length_check_error_user_long = False
    show_length_check_error_pass_short = True
    show_length_check_error_pass_long = False
    show_format_check_error_pass = False
    show_lookup_error = False

elif length_check_pass(password) == 2:
    show_presence_check_error = False
    show_length_check_error_user_short = False
    show_length_check_error_user_long = False
    show_length_check_error_pass_short = False
    show_length_check_error_pass_long = True
    show_format_check_error_pass = False
    show_lookup_error = False

# format check for password
elif not has_special_char(password):
    show_presence_check_error = False
    show_length_check_error_user_short = False
    show_length_check_error_user_long = False
    show_length_check_error_pass_short = False
    show_length_check_error_pass_long = False
    show_format_check_error_pass = True
    show_lookup_error = False
```

```
Traceback (most recent call last):
  File "C:\Users\madeb\PycharmProjects\Pythor
    elif length_check_user(username) == 1:
                           ^^^^^^^^
NameError: name 'username' is not defined
```

Unfortunately, for the register length check validation, the game says that my username variable is not defined. This is because I have not indented its definition, as well as the other 3 variables, in the elif statement for register page validation. I will need to indent them so that they can be used within the validation elif statements. I will also need to move this under the code that handles the text inputs for each input in the register page. Finally, I changed the button from reg_button to create_account_button, as this will allow an account to be registered.

```
# register page validations
if create_account_button.is_clicked(event):
    username = reg_username_input.text.strip()
    email = reg_email_input.text.strip()
    password = reg_password_input.text.strip()
    conpassword = reg_confirm_input.text.strip()

# length check for username
    if length_check_user(username) == 1:
        show_presence_check_error = False
        show_length_check_error_user_short = True
        show_length_check_error_user_long = False
        show_length_check_error_pass_short = False
        show_length_check_error_pass_long = False
        show_format_check_error_pass = False
        show_lookup_error = False
```

(insert video link)

However, my text variables are being displayed in the login page still, which hasnt been drawn on the screen. To fix this, I will make sure that the error messages are set to appear on the register page instead of the login page. Additionally, I will add the presence check error message to the register error messages as it has not already been added.

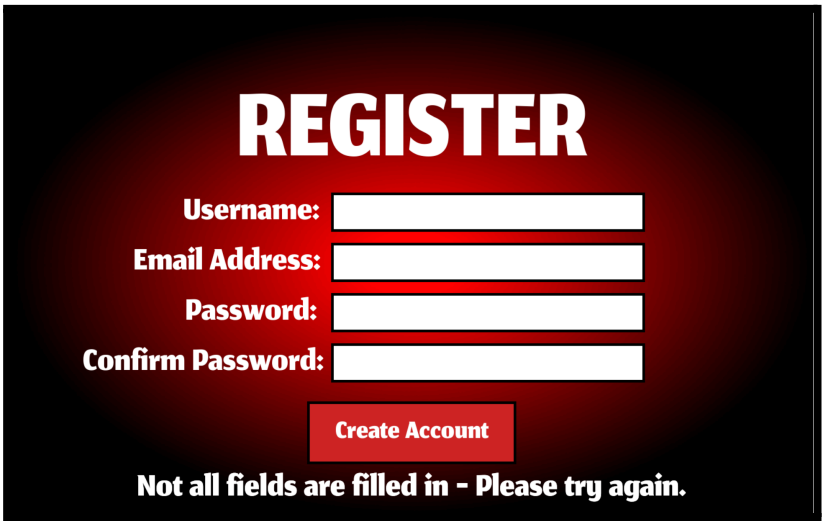
```
# presence check
    if not presence_check(username) or not presence_check(email) or not presence_check(password) or not presence_check(conpassword):
        show_presence_check_error = True
        show_length_check_error_user_short = False
        show_length_check_error_user_long = False
        show_length_check_error_pass_short = False
        show_length_check_error_pass_long = False
        show_format_check_error_pass = False
        show_lookup_error = False
```

```
#register validation variables/printing error messages
if show_presence_check_error:
    presence_check_text.draw(register_page)
elif show_length_check_error_user_short:
    length_check_text_user_short.draw(register_page)
elif show_length_check_error_user_long:
    length_check_text_user_long.draw(register_page)
elif show_length_check_error_pass_short:
    length_check_text_pass_short.draw(register_page)
elif show_length_check_error_pass_long:
    length_check_text_pass_long.draw(register_page)
elif show_format_check_error_pass:
    format_check_text.draw(register_page)
```

(insert video link) - Nothing happens after I fill in all fields because all of my current validations are successful.

From the video, it is clear that the error messages clash with the create account button, so I decided to move it lower down on the screen. However, due to risk of the messages being too low on the screen for users to see, I shifted the register title, input boxes, labels and create account button all up by 0.05.

```
#register page title
register_title = Title( text: "REGISTER",title_font, colour: "white",title_w//2,title_h * 0.25)

#register page labels and inputs
reg_username_label = Title( text: "Username:",normal_font, colour: "white",title_w * 0.3,title_h * 0.4)
reg_username_input = InputBox(title_w * 0.4,title_h * 0.36, w: 300, h: 75, text_colour: "#000000",normal_font)
reg_email_label = Title( text: "Email Address:",normal_font, colour: "white",title_w * 0.27,title_h * 0.5)
reg_email_input = InputBox(title_w * 0.4,title_h * 0.46, w: 300, h: 75, text_colour: "#000000",normal_font)
reg_password_label = Title( text: "Password:",normal_font, colour: "white",title_w * 0.3,title_h * 0.6)
reg_password_input = InputBox(title_w * 0.4,title_h * 0.56, w: 300, h: 75, text_colour: "#000000",normal_font)
reg_confirm_label = Title( text: "Confirm Password:",normal_font, colour: "white",title_w * 0.24,title_h * 0.7)
reg_confirm_input = InputBox(title_w * 0.4,title_h * 0.66, w: 300, h: 75, text_colour: "#000000",normal_font)

#create account button
create_account_button = Button(x=(title_w - 400) // 2,y=button_y * 1.55,w=400,h=120,text="Create Account",font=button_font,
    colour="#CD2626",hover_colour="#8B0000",text_colour="#FFFFFF",border_colour="#000000",border_width=5)
```



Now it was time for me to create the email format validation. I will do this by creating a function which will check every letter of the email input and check whether it contains an “@” symbol and a valid domain. As I was unsure of how to do this, I went to Stack Overflow, where a user suggested using the validate_email pip. With this in mind, I used this pip to create a function to check whether an inputted email is valid or not. I will also create its corresponding error message.

```
from validate_email import validate_email
```

```
format check for email
ef email_format(text): 1 usage
   valid_email = validate_email(text,verify=True)
   if valid_email:
       return True
   else:
       return False
ormat_email_check_text = Title( text: "Invalid email format - Please try again.",normal_font, colour: "white",title_w//2, title_h * 0.95)
```

```
elif show_format_check_error_email:
    format_email_check_text.draw(register_page)
```

```
show_format_check_error_email = False
```

```
elif not email_format(email):
    show_presence_check_error = False
    show_length_check_error_user_short = False
    show_length_check_error_user_long = False
    show_length_check_error_pass_short = False
    show_length_check_error_pass_long = False
    show_format_check_error_pass = False
    show_format_check_error_email = True
    show_lookup_error = False
```

(Insert video link)

```
Traceback (most recent call last):
  File "C:\Users\madeb\PycharmProjects\PythonProject\Jaden's NEA.py", line 389, in <module>
    elif not email_format(email):
             ~~~~~~~~~~~~^^^^^^^
  File "C:\Users\madeb\PycharmProjects\PythonProject\Jaden's NEA.py", line 278, in email_format
    valid_email = validate_email(text,verify=True)
  File "C:\Users\madeb\PycharmProjects\PythonProject\.venv\Lib\site-packages\validate_email.py", line 131, in validate_email
    raise Exception('For check the mx records or check if the email exists you must '
                    'have installed pyDNS python package')
Exception: For check the mx records or check if the email exists you must have installed pyDNS python package
```

Unfortunately, this did not work. The error message advises me to install the pip “pyDNS”, however I was unable to install it on my laptop. To combat this error, I instead decided to make sure the email address contains the basic necessary information: a username, an “@” symbol, and a domain name with a “.”. This function may be changed when I complete the verification process.

```
#format check for email
def email_format(text): 1 usage
    if "@" not in text:
        return False
    if text.count("@") != 1:
        return False
    local, domain = text.split("@")
    if not local or not domain:
        return False
    elif "." not in domain:
        return False
    else:
        return True
```

(insert video link) - The game has recognised that my email is in the correct format, hence the error message is now no longer being displayed

Match Check: My match check will ensure that the password and confirm password are the exact same as this will prevent the user from inputting any spelling errors by mistake. This will be a simple function, which will only require 2 parameters, and the check will also need an error message to notify the user that the passwords do not match.

```
#match check for password/confirm in register
def check_password_match(text1,text2):  1 usage
    if text1 != text2:
        return False
    else:
        return True
match_password_error_text = Title( text: "Passwords do not match - Please try again.",normal_font, colour: "white",title_w//2, title_h * 0.95)
```

I also created a temporary variable and error message screen to signify that the validations has been passed, which would lead on to a new account being created.

```
else:
    show_presence_check_error = False
    show_length_check_error_user_short = False
    show_length_check_error_user_long = False
    show_length_check_error_pass_short = False
    show_length_check_error_pass_long = False
    show_format_check_error_pass = False
    show_format_check_error_email = False
    show_match_check_error = False
    show_lookup_error = False
    show_create_account_successful = True
```

```
elif show_create_account_successful:
    create_account_successful_text.draw(register_page)
```

```
show_create_account_successful = False
```

(insert video link)

Lookup Check (Registration): My final validation will be making sure that the username and email entered by the user are not already within the database. I would do this by creating a similar function to my existing lookup checkfunction for the login page. I will have to create two separate functions as the username and email wont be in the same column in my database.

```
show_user_exists_error = False
show_email_exists_error = False
```

```
#lookup email check for registration
def lookup_email(text):  1 usage
    connect = sqlite3.connect("credentials.db")
    cur = connect.cursor()
    cur.execute( sql: "SELECT * FROM user_credentials WHERE email = ?", text)
    result = cur.fetchone()
    connect.close()
    if result:
        return True
    else:
        return False
email_exists_text = Title( text: "Email already exists - Please enter an alternative.",normal_font, colour: "white",title_w//2, title_h * 0.95)
```

```python
#lookup user check for registration
def lookup_user(text):  1 usage
    connect = sqlite3.connect("credentials.db")
    cur = connect.cursor()
    cur.execute( sql: "SELECT * FROM user_credentials WHERE username = ?", text)
    result = cur.fetchone()
    connect.close()
    if result:
        return True
    else:
        return False
user_exists_text = Title( text: "Username already exists - Please enter an alternative.",normal_font, colour: "white",title_w//2, title_h * 0.95)
```

```python
elif lookup_email(email):
    show_presence_check_error = False
    show_length_check_error_user_short = False
    show_length_check_error_user_long = False
    show_length_check_error_pass_short = False
    show_length_check_error_pass_long = False
    show_format_check_error_pass = False
    show_format_check_error_email = False
    show_match_check_error = False
    show_user_exists_error = False
    show_email_exists_error = True
    show_lookup_error = False
```

```python
elif lookup_user(username):
    show_presence_check_error = False
    show_length_check_error_user_short = False
    show_length_check_error_user_long = False
    show_length_check_error_pass_short = False
    show_length_check_error_pass_long = False
    show_format_check_error_pass = False
    show_format_check_error_email = False
    show_match_check_error = False
    show_user_exists_error = True
    show_email_exists_error = False
    show_lookup_error = False
```

```python
elif show_user_exists_error:
    user_exists_text.draw(register_page)
elif show_email_exists_error:
    email_exists_text.draw(register_page)
```

(Insert video link)

```
pygame 2.6.1 (SDL 2.28.4, Python 3.13.9)
Hello from the pygame community. https://www.pygame.org/contribute.html
Traceback (most recent call last):
  File "C:\Users\madeb\PycharmProjects\PythonProject\Jaden's NEA.py", line 510, in <module>
    elif lookup_user(username):
         ^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\madeb\PycharmProjects\PythonProject\Jaden's NEA.py", line 314, in lookup_user
    cur.execute("SELECT * FROM user_credentials WHERE username = ?", text)
    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
sqlite3.ProgrammingError: Incorrect number of bindings supplied. The current statement uses 1, and there are 13 supplied.
```

However, an error has occurred. This is because my parameter has been passed into the SQL statement incorrectly. The parameter must be in brackets and have a comma after, or else it will be passed in as a string and not a tuple, which will be the username row in my database.

For reference, this is what the username and email in my database is:

	Filter	Filter	Filter
1	JadenHamilton	me@gmail.com	CINCO@2025

Edited code:

```
def lookup_email(text):  1 usage
    connect = sqlite3.connect("credentials.db")
    cur = connect.cursor()
    cur.execute( sql: "SELECT * FROM user_credentials WHERE email = ?",  parameters: (text,))
```

```
#Lookup user check for registration
def lookup_user(text):  1 usage
    connect = sqlite3.connect("credentials.db")
    cur = connect.cursor()
    cur.execute( sql: "SELECT * FROM user_credentials WHERE username = ?",  parameters: (text,))
```

(insert video link)
(insert video link)

However, upon running this code, I have come across a unique error. In the first video, the email exists error message keeps appearing, even though I have changed the email address in the input box. On the other hand, in the second video, when I directly enter an email that does not exist in the database, the program registers it as a valid email straight away. When reviewing my code again, I noticed that I had missed out the show variables in the else statement, when everything entered is correct. To fix this, I added show_user_exists_error and show_email_exists_error in the else statement and set them both to false so they disappear when all validations are met.

```
else:
    show_presence_check_error = False
    show_length_check_error_user_short = False
    show_length_check_error_user_long = False
    show_length_check_error_pass_short = False
    show_length_check_error_pass_long = False
    show_format_check_error_pass = False
    show_format_check_error_email = False
    show_match_check_error = False
    show_user_exists_error = False
    show_email_exists_error = False
    show_lookup_error = False
    show_create_account_successful = True
```

(insert video link)

Maintainability

When looking at muy code, I realised that it was very sequential and not very modular, and hard for another user to read and understand despite my comments. I first started by combining all functions into one large function for registration.

```
#MAIN VALIDATION FUNCTION FOR REGISTRATION
def validate_registration(username, email, password, conpassword): 1 usage
    #presence check
    if not presence_check(username) or not presence_check(email) or not presence_check(password) or not presence_check(conpassword):
        return False, presence_check_text

    #username length
    userlen = length_check_user(username)
    if userlen == 1:
        return False, length_check_text_user_short
    if userlen == 2:
        return False, length_check_text_user_long

    #email format
    if not email_format(email):
        return False, format_email_check_text

    #password length
    passlen = length_check_pass(password)
    if passlen == 1:
        return False, length_check_text_pass_short
    elif passlen == 2:
        return False, length_check_text_pass_long
```

```
#password special character
if not has_special_char(password):
    return False, format_check_text

#matching passwords
if not check_password_match(password, conpassword):
    return False, match_password_error_text

#username exists
if lookup_user(username):
    return False, user_exists_text

#email exists
if lookup_email(email):
    return False, email_exists_text

return True, None
```

I then also created variables called is_valid and message, where the boolean of each validation will be stored, and the necessary error message.

```
# register page validations
if create_account_button.is_clicked(event):
    username = reg_username_input.text.strip()
    email = reg_email_input.text.strip()
    password = reg_password_input.text.strip()
    conpassword = reg_confirm_input.text.strip()

    is_valid, message = validate_registration(username, email, password, conpassword)
```

Next, I replaced all the error message flags with simple if statement which will use the message variable to determine which error message needs to be drawn. This will save me from being confused with each of the variables.

```
# error message to print/success
if not is_valid:
    current_register_error = message
else:
    current_register_error = create_account_successful_text
```

```
#validation state variables
current_register_error = None
```

The final step to ensuring this change works is by changing the output statements so that the current error is drawn. For now this will include the account created successfully page.

```
#register validation  - printing error messages
if current_register_error:
    current_register_error.draw(register_page)
```

I also realised that my presence check does not return a boolean but returns a string, so I also went back and edited this.

```
#presence check function
def presence_check(text):  6 usages
    return text.strip() != ""
```

Here is the output of the new registration page:
(insert video link)

Next, I went back on my Login page and repeated the same steps I did on the Register page. I started by creating the function.

```
#MAIN VALIDATION FUNCTION FOR LOGIN
def validate_login(username,password):
    #presence check
    if not presence_check(username) or not presence_check(password):
        return False, presence_check_text
    #lookup check username
    if lookup_user(username):
        return False, user_exists_text

    return True, login_successful_text
```

After this, I created another is_valid and message variable, which will determine whether the input is valid or not, and the corresponding error to output, based on the function.

```
#data validation for login
if login_page_button.is_clicked(event):
    username = login_username_input.text.strip()
    password = login_password_input.text.strip()

    is_valid, message = validate_login(username,password)
```

I then created a variable called current_login_error, which will store the message that needs to be outputted based on the error. I then included this in an if statement underneath the login validation.

```
#validation state variables
current_login_error = None
```

```
# error message to print/success
if not is_valid:
    current_login_error = message
else:
    current_login_error = login_successful_text
```

Finally, I created another if statement, which will draw the error message on the screen, based on the error that the user has caused in the program. Just like teh register page, the success text will be temporary to signify that all validations have been passed.

```
#login validation - printing error messages
if current_login_error:
    current_login_error.draw(login_page)
```

Here is the output of the code:()

Upon running the program, the code prints over the error unexpectedly. I reviewed my code again and I found out that I had used the lookup_user function instead of the check_user function. I also put the wrong error message. I changed this below, and the code now runs as intended.

```
#MAIN VALIDATION FUNCTION FOR LOGIN
def validate_login(username,password):  1 usage
    #presence check
    if not presence_check(username) or not presence_check(password):
        return False, presence_check_text
    #lookup check username
    if not check_User(username,password):
        return False, lookup_check_text

    return True, login_successful_text
```

(insert video link)

Reset Password Page

This will be the final validation I complete for my login page and Prototype 1. This page will be different from the login and registration validations as I plan to use the email inputted by the user to send them a one time code, to verify the code sent. This will then require a new page to be created separate from the reset password page, so that the user can input the one time code, which will securely set the new password.

Firstly, I created the main function to validate the reset password inputs, and created the temporary message to display the successful password reset.

```
#MAIN VALIDATION FUNCTION FOR RESET PASSWORD
def validate_reset_password(email, password, conpassword): 1 usage
    #presence check
    if not presence_check(email) or not presence_check(password) or not presence_check(conpassword):
        return False, presence_check_text

    #email format
    if not email_format(email):
        return False, format_email_check_text

    #email exists
    if lookup_email(email):
        return False, email_exists_text

    #matching passwords
    if not check_password_match(password, conpassword):
        return False, match_password_error_text

    return True, None
```

```
reset_pass_successful_text = Title( text: "Password reset successfully!",normal_font, colour: "white",title_w//2, title_h * 0.95)
```

Next, I created the variable to store the current error message for the reset password page. I also created the if statement to signify the code which will occur when the reset password button is pressed, and assigned each input from the input box using variables. Each variable has text.strip() to remove accidental spaces by the user.

```
current_reset_password_error = None
```

```
if reset_button.is_clicked(event):
    email = reset_email_input.text.strip()
    password = reset_password_input.text.strip()
    conpassword = reset_confirm_input.text.strip()
```

Underneath this block of code, I added the same variables is_valid and message, to store the current error message to be displayed. Then, I added another if statement to draw these statements on the screen for the user to see.

```
is_valid, message = validate_reset_password(email, password, conpassword)

if not is_valid:
    current_reset_password_error = message
else:
    current_register_error = reset_pass_successful_text
```

```
#reset validation - printing error messages/success
if current_reset_password_error:
    current_reset_password_error.draw(reset_page)
```

Upon reflection of the function, the error message I used for the email lookup is to show that the email already exists, but I want one that says that the email does not exist. To fix this, I created a separate error message just underneath the lookup email function.

```
email_exists_text = Title( text: "Email already exists - Please enter an alternative.",normal_font, colour: "white",title_w//2, title_h * 0.95)
email_not_exists_text = Title( text: "Email not found - Please try again.",normal_font, colour: "white",title_w//2, title_h * 0.95)
```

```
#lookup email
if not lookup_email(email):
    return False, email_not_exists_text
```

I also realised that I had left out the length checks and special character checks for the password, so I included those.

```
#password length
passlen = length_check_pass(password)
if passlen == 1:
    return False, length_check_text_pass_short
elif passlen == 2:
    return False, length_check_text_pass_long

#password special character
if not has_special_char(password):
    return False, format_check_text
```

Additionally, I had assigned the wrong variable in my is_valid event loop. I used current_register_error instead of current_reset_password_error, which means it would never print “Password reset successfully!”. I changed this below:

```
if not is_valid:
    current_reset_password_error = message
else:
    current_reset_password_error = reset_pass_successful_text
```

This is the output of this code: (video link)

Email verification for Reset Password Page

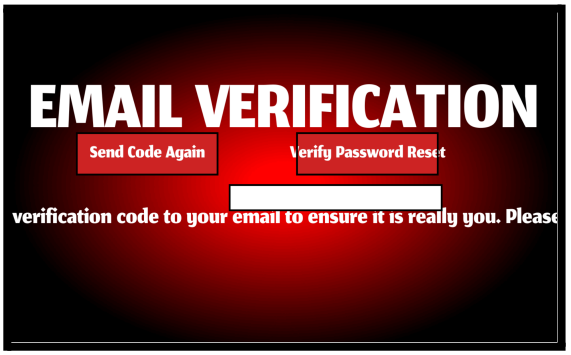
To finish off this section, following the reset password page, I will create a section to send an email to a user with a temporary one time code to verify their account.

To start off, I created a new page called Email verification. This page will contain a label for the heading, an input box to enter the code, and buttons to send the code and validate email for the reset password.

```
#email verification page creation
email_veri_page = pygame.display.set_mode( size: (title_w,title_h), pygame.FULLSCREEN | pygame.SCALED)
email_veri_bg = pygame.transform.smoothscale(titlebg,  size: (title_w,title_h))

#email verification page title
email_veri_page_title = Title( text: "EMAIL VERIFICATION",title_font, colour: "white",title_w//2,title_h * 0.3)

#email verification page label and input
code_sent_text = Title( text: "We have successfully sent a one-time verification code to your email to ensure it is really you. Please enter the code in the text box below:",
                  normal_font, colour: "white",title_w//2,title_h * 0.6)
code_input = InputBox(title_w * 0.4,title_h * 0.5, w: 300, h: 75, text_colour: "#000000",input_font)

#email verification page buttons
send_code_button = Button(x=button_x * 0.6// 2,y=button_y * 0.7,w=400,h=120,text="Send Code Again",font=button_font,
    colour="#CD2626",hover_colour="#8B0000",text_colour="#FFFFFF",border_colour="#000000",border_width=5)

verify_reset_button = Button(x=button_x * 1.3,y=button_y * 0.7,w=400,h=120,text="Verify Password Reset",font=button_font,
    colour="#CD2626",hover_colour="#8B0000",text_colour="#FFFFFF",border_colour="#000000",border_width=5)
```

I wrote the code to display the text from the text box, as well as the text box itself, and all the other features needed on the screen. I tested the code to see what the output would show:

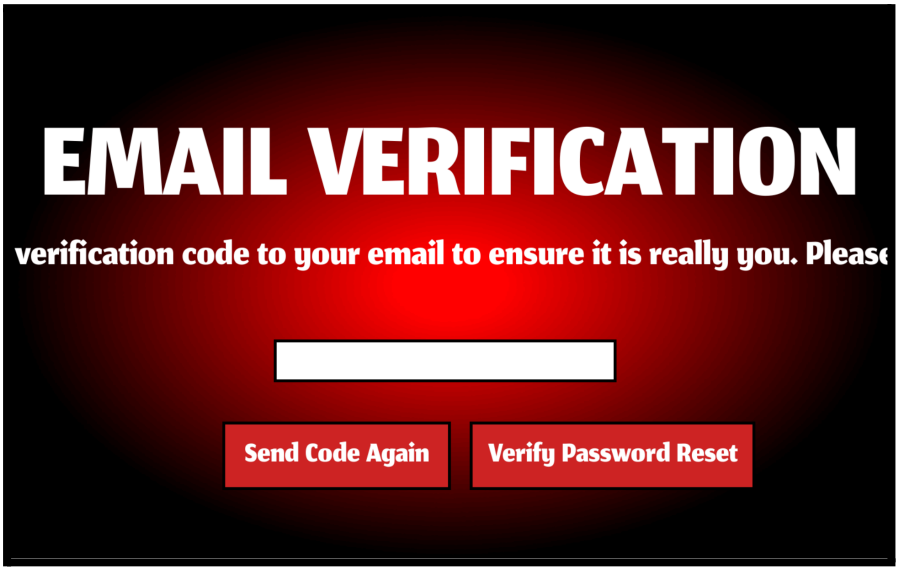
```
    #verify email text
    code_input.do_event(event)

#drawing verify email page labels/buttons/input
email_veri_page.blit(email_veri_bg,  dest: (0,0))
email_veri_page_title.draw(email_veri_page)
code_sent_text.draw(email_veri_page)
code_input.draw(email_veri_page)
send_code_button.draw(email_veri_page)
verify_reset_button.draw(email_veri_page)
```



The code has shown many issues that I had made, which I did not expect to make. Here is a list:

- The buttons are way too far up the page. I need to adjust the y-coordinate of the buttons so they sit below the other features.
- The message to signify the verification code has been sent does not fit on the page, and it is also vertically centered. To fix this I will need to add new lines in the text so that all the text can be shown on the single screen. I will also need to adjust the y-coordinate of the code so that it is higher up on the page.
- The text on the verify password reset button extends over the button. To fix this, I will need to make this specific button longer, which will require me increasing the value of the button's width
- The text box is not centered, and is too high on the screen. I will need to adjust both the x-coordinate and y-coordinate so it is put in the right place. I will also shorten the width of the text box as there will not be a lot of text for the user to input.

```
#email verification page label and input
code_sent_text = Title( text: "We have successfully sent a one-time verification code "
                        "to your email to ensure it is really you. Please enter the "
                        "code in the text box below:",
                        normal_font, colour: "white",title_w//2,title_h * 0.45)
code_input = InputBox(title_w * 0.3,title_h * 0.6, w: 59, h: 75, text_colour: "#000000",input_font)

#email verification page buttons
send_code_button = Button(x=button_x * 0.6,y=button_y * 1.5,w=400,h=120,text="Send Code Again",font=button_font,
    colour="#CD2626",hover_colour="#8B0000",text_colour="#FFFFFF",border_colour="#000000",border_width=5)

verify_reset_button = Button(x=button_x * 1.3,y=button_y * 1.5,w=500,h=120,text="Verify Password Reset",font=button_font,
    colour="#CD2626",hover_colour="#8B0000",text_colour="#FFFFFF",border_colour="#000000",border_width=5)
```



Unfortunately, the text is still not fitting on the entire page, and also the text box has not reduced in width. When I looked back at my input box function, I realised that the width was being set to 600 by default, instead of what the user is inputting. As this would affect all input boxes, I went and changed 600 to rect.w, which will be the user input for tht width, and changed every existing text box's width to 600, while changing the code input box to 300.

```
textsurface = self.font.render(self.text, True, self.text_colour)
width = max(self.rect.w,textsurface.get_width()+10)
self.inputBox.w = width
```

```
code_input = InputBox(title_w * 0.4,title_h * 0.6, w: 300, h: 75, text_colour: "#000000",input_font)
```

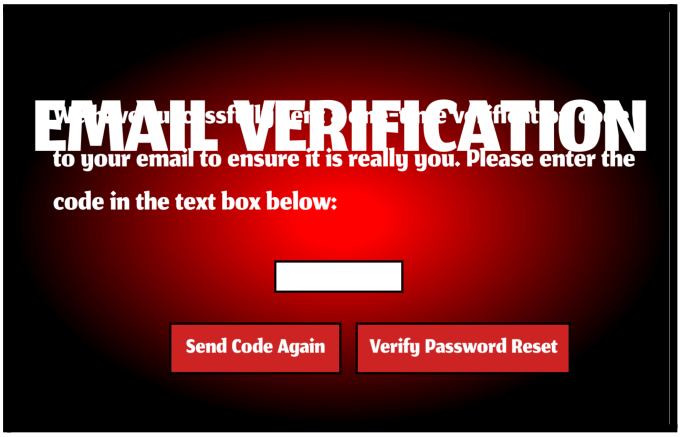

```
#email verification page label and input
code_sent_text = Title( text: """
We have successfully sent a one-time verification code
to your email to ensure it is really you. Please enter the
code in the text box below:
""",
```

I then went to fix the issue of the text not showing on the entire screen. I tried using triple brackets to show that the text goes over one line. Here is the output:

Unfortunately, this still did not work. It seems my title class cannot display multiple lines in paragraph format. To fix this I went to stack overflow and found a function I can use. To implementthis, I will need to set the text to its own unique variable and pass it in as text. The function splits the text into lines and are all drawn separately using a for loop, based on the font, surface, and colour.

```
#render text over multiple lines
def render_multi_lines(surface,text,x,y,font,colour):
    lines = text.splitlines()
    line_height = font.get_linesize()
    for index,line in enumerate(lines):
        surface.blit(font.render(line,True,colour),(x,y + (index * line_height)))
```

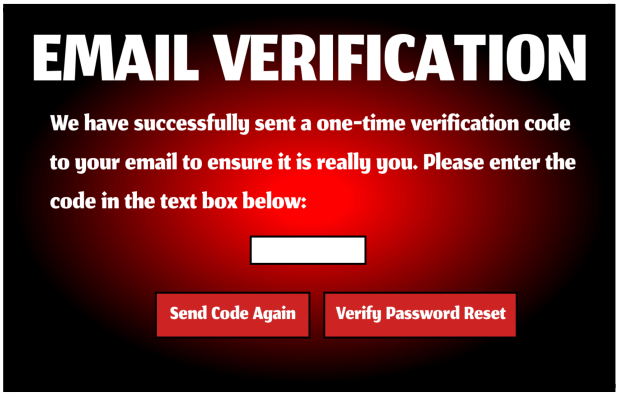
```
#drawing verify email page labels/buttons/input
email_veri_page.blit(email_veri_bg, dest: (0,0))
email_veri_page_title.draw(email_veri_page)
render_multi_lines(email_veri_page, code_sent_message, x: 100,title_h * 0.1,normal_font, colour: "#FFFFFF")
code_input.draw(email_veri_page)
send_code_button.draw(email_veri_page)
verify_reset_button.draw(email_veri_page)
```



Thishas worked, however it overlaps with the email verification title. I fixed this by moving the title above the message, and moving the message slightly lower, so that both pieces of text can be seen.

```
render_multi_lines(email_veri_page, code_sent_message, x: 100,title_h * 0.15,normal_font, colour: "#FFFFFF")
```

```
#email verification page title
email_veri_page_title = Title( text: "EMAIL VERIFICATION",title_font, colour: "white",title_w//2,title_h * 0.15)
```



Next, I moved on to displaying the errors for the page. This page will only need a message for a presence check, and if the code matches the one sent, so I reused the presence check message, and created a new error message called match_code_error_text, to display if the codes do not match. To demonstrate the code working correctly, I also created a temporary variable called sent_code to use within the function. I also renamed my match vheck variable, so that it can be used universally throughout the program, not just for the password.

```
#code check for password reset
sent_code = " "
match_code_error_text = Title( text: "Code does not match - Please try again.",normal_font, colour: "white",title_w//2, title_h * 0.95)
```

```
#VALIDATION FUNCTION FOR PASSWORD RESET:
def validate_reset_pass_verify(code):
    #presence check
    if not presence_check(code):
        return False, presence_check_text
    if not check_match(code,sent_code):
        return False, match_code_error_text

    return True, None
```

```
def check_match(text1,text2):  3 usages
```

I then created is_vailid and message variables to store whether the user input is valid or not, and the corresponding message to display. This message is then drawn on the screen using an if statement.

```
if verify_reset_button.is_clicked(event):
    code = code_input.text.strip()

    is_valid, message = validate_reset_pass_verify(code)

    if not is_valid:
        current_email_verify_error = message
```

```
if current_email_verify_error:
    current_email_verify_error.draw(email_veri_page)
```

(insert video link)

Now that the validations are working, before I make the function to send an email to the user, I created a simple function called generate_code, to generate a random 6-digit number. I will use this within the email, for the user to then input in the game.

```python
#generate code
def generate_code():  1 usage
    return random.randint( a: 100000, b: 999999)

codecode = generate_code()
print(codecode)
```

```
551113
```

Now I moved on to creating the send_email function. As I did not know how to use smtplib, I watched BroCode's video on sending emails in python, to assist me. Using his structure, I attempted to make a valid function that will send an email to my personal account. I added print statements to confirm whether an email was sent or was not sent. Additionally, I had to create an App password, for the sender account. This is a temporary 16-digit password that the program can use to access my account, then send the email to my personal account. I tried this code below:

```python
#verification email
def send_email(email,code):  1 usage
    sender = "cincoofficialgame@gmail.com"
    receiver = email
    password = "ffedtvockrkkrucn"
    subject = "Email Verification to Reset Password"
    body = f"Here is your code, please enter it in the text box within the game: \n {code}"
    message = f"""From: {sender}
To: {receiver}
Subject: {subject}

{body}
"""
    server = smtplib.SMTP( host: 'smtp.gmail.com',  port: 587)
    server.starttls()

    try:
        server.login(sender,password)
        server.sendmail(sender, receiver, message)
        server.quit()
        print("Code sent successfully")
    except smtplib.SMTPAuthenticationError:
        print("Error")

jadenemail = "jadenhamil08@gmail.com"
send_email(jadenemail,codecode)
```

However, this did not work, and when trying more than once, the sender account was temporarily disabled for 48 hours. I then researched new methods to allow this function to work. I came across monkey see monkey do's video, where he used a function called os.getenv() to get his password. I then researched this function and found out it recalls an environment variable within my operating system. I decided then to create an app password and assign it as an environment variable using windows powershell, and recall this variable using os.getenv() and set this to the password variable in my code. I will also need to import os, as it is a library I have not already used within my code yet. The password has been hidden from the command prompt statement for security reasons.

```
PS C:\Windows\System32> $env:GMAIL_APP_PASSWORD="
```

```
#importing libraries
import os
```

```
sender = "cincoofficialgame@gmail.com"
receiver = email
password = os.getenv("GMAIL_APP_PASSWORD")
```

```
581072
Traceback (most recent call last):
  File "C:\Users\madeb\PycharmProjects\PythonProject\Jaden's NEA.py", line 514, in <module>
    send_email(jadenemail,codecode)
    ~~~~~~~~~~^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\madeb\PycharmProjects\PythonProject\Jaden's NEA.py", line 502, in send_email
    server = smtplib.SMTP('smtp.gmail.com', 587)
  File "C:\Users\madeb\AppData\Local\Programs\Python\Python313\Lib\smtplib.py", line 264, in __init__
    (code, msg) = self.connect(host, port)
                  ~~~~~~~~~~~~^^^^^^^^^^^^
  File "C:\Users\madeb\AppData\Local\Programs\Python\Python313\Lib\smtplib.py", line 350, in connect
    self.sock = self._get_socket(host, port, self.timeout)
                ~~~~~~~~~~~~~~~~^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\madeb\AppData\Local\Programs\Python\Python313\Lib\smtplib.py", line 321, in _get_socket
    return socket.create_connection((host, port), timeout,
           ~~~~~~~~~~~~~~~~~~~~~~~~~^^^^^^^^^^^^^^^^^^^^^^
                                    self.source_address)
                                    ^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\madeb\AppData\Local\Programs\Python\Python313\Lib\socket.py", line 864, in create_connection
    raise exceptions[0]
  File "C:\Users\madeb\AppData\Local\Programs\Python\Python313\Lib\socket.py", line 849, in create_connection
    sock.connect(sa)
    ~~~~~~~~~~~~~^^^^
```

Unfortunately, another error has appeared in my code, this time the code states that the connection attempt failed because the connected party did njot respond properly after a period of time or the host failed to respond. When researching this error, I found out that this was a network issue, mainly linked to my school's firewall blocking the conenction. I decided to test the code when I arrived at home. However, this message appeared.

```
'NoneType' object has no attribute 'encode'
```

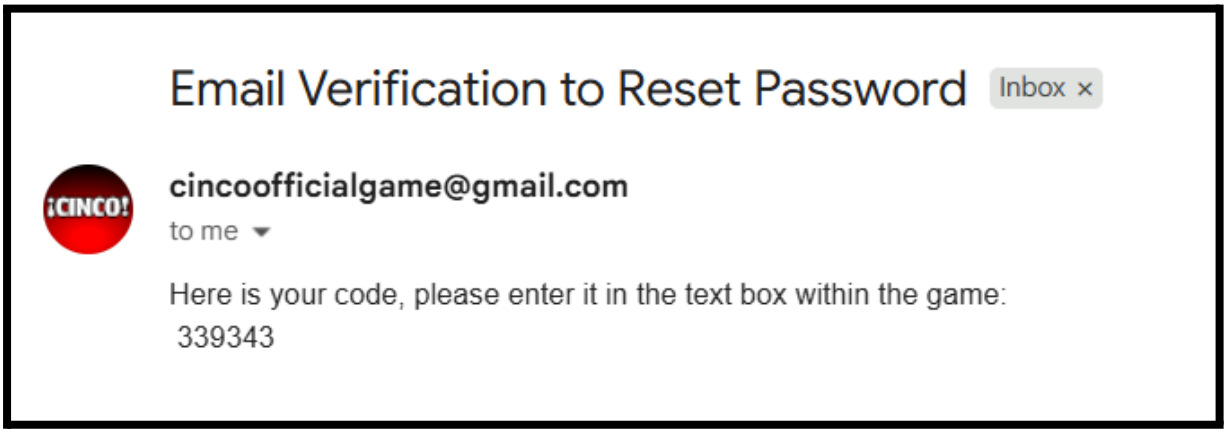
I discovered that the password was not being stored using this function. Instead I watched a video by Dave Ebbalaar, who explained how to use environment variables in python. I created a file called .env, and downloaded the package dotenv. In the file, I placed a variable called APP_PASSWORD and gave it the value of a new app password I had created for the gmail account. Next, I went to the main code, and wrote lines of code to find the .env file and retrieve the value of APP_PASSWORD securely. Here is the output of this method below:

```
(.venv) PS C:\Users\madeb\PycharmProjects\PythonProject> pip install dotenv
Collecting dotenv
  Downloading dotenv-0.9.9-py2.py3-none-any.whl.metadata (279 bytes)
Collecting python-dotenv (from dotenv)
  Downloading python_dotenv-1.2.1-py3-none-any.whl.metadata (25 kB)
Downloading dotenv-0.9.9-py2.py3-none-any.whl (1.9 kB)
Downloading python_dotenv-1.2.1-py3-none-any.whl (21 kB)
Installing collected packages: python-dotenv, dotenv
Successfully installed dotenv-0.9.9 python-dotenv-1.2.1
```

```
dotenv_path = find_dotenv()
load_dotenv(dotenv_path)
password = os.getenv("APP_PASSWORD")
```

```
339343
Code sent successfully
```

(insert video link)



Next, I moved on to making sure that the code entered will match when entered by the user. I first added the sent_code parameter to the code, as it would be needed to compare the two codes. I also made sure that the successful message would display if both passwords matched. However, as an email hasn't been inputted, and I won't be able to do this until all pages have been linked to one another, I used my personal email just to test. I renamed the

“send code again” button to only “send code” and thus edited the page message, as it will be more convenient for the user, and they will know when the code has been sent. Finally, to make the sent_code variable present in both if statement, I created a variable to store the current code that has been sent, so that it can be used globally, and converted the code into a string so that it won’t clash with user input.

```
#VALIDATION FUNCTION FOR PASSWORD RESET:
def validate_reset_pass_verify(code,sent_code):  1 usage
    #presence check
```

```
current_code_sent = " "
```

```
if send_code_button.is_clicked(event):
    current_code_sent = str(generate_code())
    send_email(jadenemail, current_code_sent)

if verify_reset_button.is_clicked(event):
    code = code_input.text.strip()

    is_valid, message = validate_reset_pass_verify(code,current_code_sent)
```

```
if not is_valid:
    current_email_verify_error = message
else:
    current_email_verify_error = reset_pass_successful_text
```

(insert video link)

Button Functionality/Flow

This is the final section of my prototype 1. Here, I will be focusing on making sure each button directs to different pages, allowing my game to flow.

Pages: To start, I firstly put each block of code to draw each page in its own subroutine. I decided it would be best to have a variable which sets the current page, while putting every page in its own class to draw, and handle events, such as buttons being clicked. I did this by creating a parent class called Page(), with the functions draw_page() and handle_event(). I then used inheritance to create all of the pages I have used so far, including the menu page.

```
class Page:  6 usages
    def draw_page(self,surface):  6 usages
        surface.blit(bg,(0,0))

    def handle_event(self,event):
        pass
```

Resolutions: To prevent myself from recreating multiple screens to display like I had before, I created a universal variable called screen, which will represent the background, so that all the features of each page will be displayed on the same background, while keeping the background the same. Here I edited the code from earlier, so that the titlebg is renamed to bg, as it is used universally, and made sure that pygame chooses the right resolutions to use. Before, I had found the resolution of the screen being used by the user, but by using pygame.FULLSCREEN | pygame.SCALED, pygame can choose its own internal resolution and ignore my parameters. I reordered the code so that pygame chooses the correct resolution and it is used within the game.

```
#screen creation
screen = pygame.display.set_mode( size: (title_w,title_h), pygame.FULLSCREEN | pygame.SCALED)

#get screen size
title_w, title_h = screen.get_size()

#load original image / scale page
bg = pygame.image.load("Frontpage.png").convert()
bg = pygame.transform.smoothscale(bg,  size: (title_w, title_h))
```

Within each child class from the parent class Page, I moved everything that had been defined earlier within the code, such as labels, inputs and buttons, into their own designated class, within the initialise function. For example, for the title page, I moved the main title, login, register and exit buttons into the initialise function. Within the handle_event function, I will put conditions, such as when letters are being typed, and for validations. For example, in the login page, I will put the statements to show inputs for the username and password, and the validations to check whether credentials are valid and if anything has been entered. I will also put if statements to direct between each page. Here is the title and login page class, showing the framework that I will repeat for the other pages:

```python
# Title Page Class
class TitlePage(Page):  3 usages
    def __init__(self):
        # title creation(main)
        self.title_main = Title( text: "CINCO!", title_font,  colour: "white", title_w // 2, title_h * 0.3)

        # title page buttons
        self.login_button = Button(x=button_x, y=button_y, w=300, h=120, text="LOGIN",
            font=button_font,colour="#CD2626", hover_colour="#8B0000", text_colour="#FFFFFF", border_colour="#000000",border_width=5)

        self.reg_button = Button(x=button_x, y=button_y * 1.4, w=300, h=120, text="REGISTER",
            font=button_font,colour="#CD2626", hover_colour="#8B0000", text_colour="#FFFFFF", border_colour="#000000",border_width=5)

        self.exit_button = Button(x=button_x * 0.15, y=button_y * 1.75, w=150, h=100, text="EXIT",
            font=button_font,colour="#CD2626", hover_colour="#8B0000", text_colour="#FFFFFF", border_colour="#000000",border_width=5)

    def handle_event(self,event):  1 usage
        global current_page
        if self.login_button.is_clicked(event):
            current_page = LoginPage()
        if self.reg_button.is_clicked(event):
            current_page = RegisterPage()
        if self.exit_button.is_clicked(event):
            pygame.quit()

    def draw_page(self,surface):  1 usage
        super().draw_page(surface)
        self.title_main.draw(surface)
        self.login_button.draw(surface)
        self.reg_button.draw(surface)
        self.exit_button.draw(surface)
```

```python
# Login Page Class
class LoginPage(Page):  4 usages
    def __init__(self):
        # login page title
        self.login_username_input = InputBox(title_w * 0.4,title_h * 0.51, w: 600, h: 70, text_colour: "#000000",input_font)
        self.login_password_input = InputBox(title_w * 0.4,title_h * 0.61, w: 600, h: 70, text_colour: "#000000",input_font)
        self.login_title = Title( text: "LOGIN", title_font,  colour: "white", title_w // 2, title_h * 0.3)

        # login labels
        self.login_username_label = Title( text: "Username:", normal_font,  colour: "white", title_w * 0.3, title_h * 0.55)
        self.login_password_label = Title( text: "Password:", normal_font,  colour: "white", title_w * 0.3, title_h * 0.65)

        # login page buttons
        self.login_page_button = Button(x=button_x * 0.6, y=button_y * 1.5, w=300, h=120, text="Login",
            font=button_font,colour="#CD2626", hover_colour="#8B0000", text_colour="#FFFFFF",border_colour="#000000", border_width=5)
        self.forgot_password_button = Button(x=button_x * 1.3, y=button_y * 1.5, w=400, h=120, text="Forgot Password",
            font=button_font,colour="#CD2626", hover_colour="#8B0000", text_colour="#FFFFFF",border_colour="#000000", border_width=5)
        self.back_button = Button(x=button_x * 0.15, y=button_y * 1.75, w=150, h=100, text="BACK",
            font=button_font,colour="#CD2626", hover_colour="#8B0000", text_colour="#FFFFFF", border_colour="#000000",border_width=5)

    def handle_event(self,event):
        global current_page,current_error_message
        self.login_username_input.do_event(event)
        self.login_password_input.do_event(event)

        # data validation for login
        if self.login_page_button.is_clicked(event):
            username = self.login_username_input.text.strip()
            password = self.login_password_input.text.strip()
            is_valid, message = validate_login(username, password)

            # error message to print/success
            if not is_valid:
                current_error_message = message
            else:
                current_error_message = None
                current_page = MenuPage()

        elif self.forgot_password_button.is_clicked(event):
            current_page = ResetPassPage()
        elif self.back_button.is_clicked(event):
            current_page = TitlePage()
```

```
def draw_page(self,surface):
    super().draw_page(surface)
    self.login_title.draw(surface)
    self.login_username_label.draw(surface)
    self.login_password_label.draw(surface)
    self.login_username_input.draw(surface)
    self.login_password_input.draw(surface)
    self.login_page_button.draw(surface)
    self.forgot_password_button.draw(surface)
    self.back_button.draw(surface)
```

Current variables: In the handle_event function, I have included the global variable current_page. I will store the current page that is being displayed, to allow pages to be quickly switched between. In order to do this, I also defined this just before the gameloop, and set it to the title page, the first page to be shown on the screen. I also created variables, current_email to store the email address being used to reset a password for the reset password and email verification pages, and current_code_sent to store the code that will be sent to the user's email address on the email verification page.

```
while runtime:
    for event in pygame.event.get():
        current_page.handle_event(event)

    current_page.draw_page(screen)
    pygame.display.flip()
```

GameLoop: In the gameloop, I removed the previous statement to quit pygame, as this will now be done using the exit buttons. I put the handle event function and draw page functions within the gameloop, so they will execute based on the page which is currently being displayed.

Exit/Back Buttons: Within each class, I added either an exit or back button, to allow the user to exit each page.

Here is the output of all of the code changes: (insert video link)

In the video shown, I purposefully made errors within the login section (lookup check) and the register section (presence check), but error messages did not show up, as I had forgotten to write the if statements to display them.

Instead of keeping each individual variable to hold the error messages based on the pages, I created a single variable to store every single variable. I also made sure that when each error message has been fixed and the buttons change page, I will return none to the current error variable, so that the error message does not display onto the next page. This variable I create will need to be made global within the handle_event function within each class so that it changes outside the class and for the entire game. I made sure to include an if statement in the game loop to display the corresponding error message.

```
current_error_message = None
```

```
if current_error_message:
    current_error_message.draw(screen)
```

Here's how it would look in the register page class:

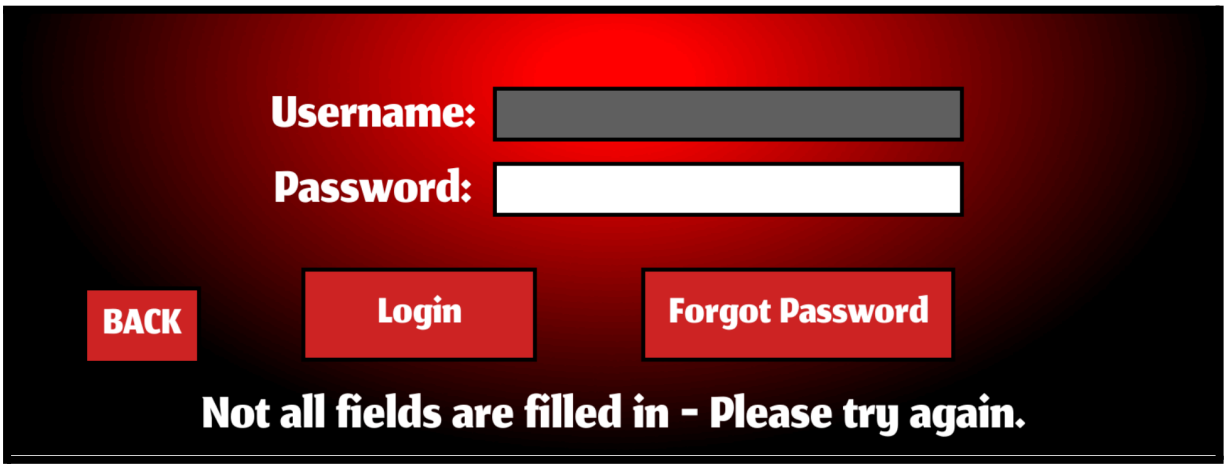
```
def handle_event(self,event):
    global current_page,current_error_message
```

```
    # error message to print/success
    if not is_valid:
        current_error_message = message
    else:
        current_error_message = None
        current_page = LoginPage()
```

Here is the output of me retrying the errors in the login page: (insert video link)

The errors are now displaying correctly, and disappearing when switching pages, however, they overlap with the exit and back buttons. To fix this, I moved both buttons slightly higher, so that the error messages can be visibly seen.

```
, y=button_y * 1.55, w=
r_colour="#8B0000", tex
```



To finish, I went back on every button and made sure they all did their functions:

Create account: Previously, I created a function called addUser to add a user to the database. To do this, in the if statement when the create button is clicked, I will place the function, using the user inputs as parameters.

```
# error message to print/success
if not is_valid:
    current_error_message = message
else:
    addUser(username,email,password)
    current_error_message = None
    current_page = LoginPage()
```

(Insert video link)

	username	email	password
	Filter	Filter	Filter
1	JadenHamilton	me@gmail.com	CINCO@2025
2	jadenhamilton	jadenhamil08@gmail.com	abababab@

Password update after reset: To do this I will need to create a function to change data within the database, specifically the password. Using the current email variable, I will store the email entered in the reset password page. I will then include the change password function after the verify password reset button is pressed. I will also need a current password variable, just like the current email, to store the password that will be changed.

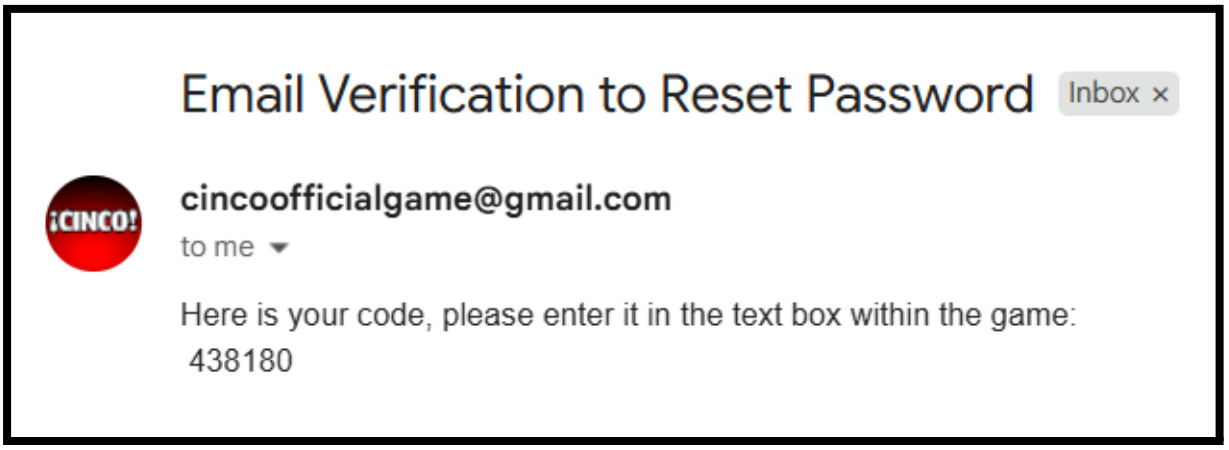
```
if not is_valid:
    current_error_message = message
else:
    current_error_message = None
    current_email = email
    current_page = EmailVeriPage()
```

```
# change password in the database
def changePassword(email,password):
    connect = sqlite3.connect("credentials.db")
    cur = connect.cursor()
    cur.execute( sql: "UPDATE user_credentials SET password=? WHERE email=?", parameters: (password,email))
    connect.commit()
    connect.close()
```

```
if not is_valid:
    current_error_message = message
else:
    changePassword(current_email,current_password)
    current_error_message = None
    current_page = LoginPage()
```

```
current_password = " "
```

(Insert video link)



Despite entering the correct code, the game says that the codes do not match. I reviewed my code and realised that I needed to convert the user input into a string, just like I did before when comparing the codes. Additionally, I saw that the current password variable does not change when the reset password button is pressed. I added this to the if statement, and made the variable global.

```
global current_page,current_error_message,current_email,current_password
```

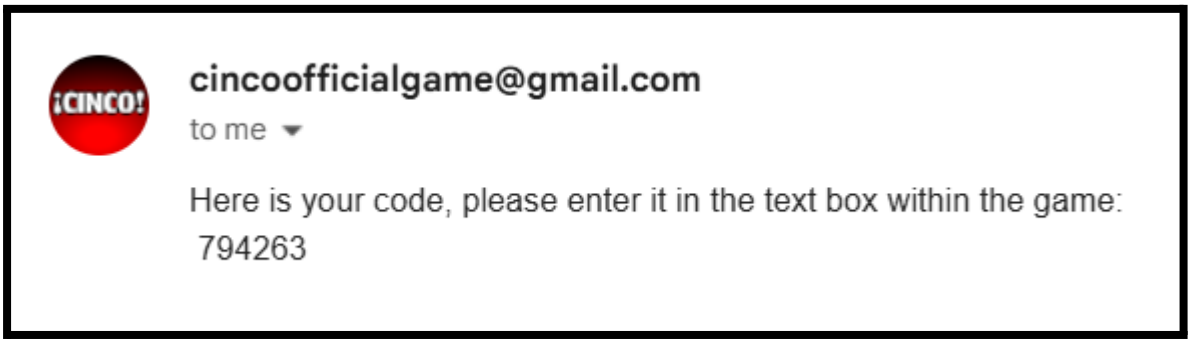
```
else:
    current_password = password
```

```
if self.send_code_button.is_clicked(event):
    current_code_sent = str(generate_code())
    send_email(current_email,current_code_sent)
```

```
Traceback (most recent call last):
  File "C:\Users\madeb\PycharmProjects\PythonProject\Jaden's NEA.py", line 713, in <module>
    current_page.handle_event(event)
    ~~~~~~~~~~~~~~~~~~~~~~~~~~~~^^^^^^^
  File "C:\Users\madeb\PycharmProjects\PythonProject\Jaden's NEA.py", line 649, in handle_event
    changePassword(current_email,current_password)
    ~~~~~~~~~~~~~~^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\madeb\PycharmProjects\PythonProject\Jaden's NEA.py", line 155, in changePassword
    cur.execute("UPDATE user_credentials SET password=? WHERE email=?",(password,email))
    ~~~~~~~~~~~^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
sqlite3.OperationalError: database is locked
```

This error came up when running the code. I realised that this was because I had kept the database open while running the code, which would then not allow the game to access the database. I closed DB browser and retried the code.

(insert video link) - Password has changed, database updated as access has been granted in the login page.



2	jadenhamilton	jadenhamil08@gmail.com	abcdabcd@

Password has changed from “abababab@” to “abcdabcd@”.

Post-Development Testing

Evaluation