KMK Fighter 2000

Video:
https://drive.google.com/file/d/10uso8Ix5dxvkyXGAAt6j1qlMucqA3-jd/view?usp=share_link

CENTRE ID : 57117
CANDIDATE NAME: KIRAN MAHESH KUMAR
CANDIDATE : 6143

ANALYSIS

Problem at hand

At School C, the Gaming Club was once a hub of activity, bringing students together to play board games and video games. But due to the pandemic, the club was closed for a while, and when the club reopened, the enthusiasm wasn't quite the same. The students were playing in smaller groups, and the overall atmosphere could have been more lively and exciting.

This provides me with an opportunity to revive the gaming club and bring back the sense of community it once had. With some effort and innovation, I believe I can make this happen. By working together with members of the club, I can recreate the fun and engaging atmosphere the club once had, with a game.

Furthermore, the lack of gaming tournaments at School C is also a problem that I am determined to solve. My solution is to introduce a tournament system with the added excitement of prizes for participants. My choice of game will be carefully thought out. The game will be locally based, meaning players can only get better by competing against each other by coming to the club. This will not only improve their skills but also help increase the popularity of the club. "

My proposal is going to be a combat game. I feel this will maintain a simple goal: defeat the opponent. And with the many characters, fighting combinations and overall competitive and entertaining environment I feel it would be a great genre for my game to be based on.

Furthermore, let's not forget the social aspect of these clubs as the students will get to meet new people and have a great time.

Analysis of existing games with pros and cons:

Brawlhalla

Brawlhalla is a free 2D fighter game made in 2014. It includes many single-player, multiplayer, and co-op modes and with a variety of brawlers to choose from.

Pros:

- Free-to-play: This one-of-a-kind game is actually free on every platform definitely giving it an advantage over other popular fighting game franchises such as Super Smash Bros and Street Fighter

- Easy to learn, hard to master: While brawlhalla's controls and a general understanding of the modes are easy to pick up, however, mastering the timing and controls of your attacks, movements and defensive strategies will take practice. This depth in gameplay makes players come back for more

- Huge Roster of characters: Countless characters the players can choose from all with unique abilities, weapons and playstyles. This gives the game diversity in how the game is played

- Cross Platform and free: Brawlhalla is free and supported on multiple platforms such as PC, Xbox, Playstation etc giving it significantly bigger access and opportunity to new audiences

- Very good graphics, UI, overall pleasing experience for the user which I believe will be vital for the presentation and appeal to my audience

Cons:

- Repetitive Gameplay: With the characters and game modes lacking in variety, the gameplay becomes rather boring and repetitive.

- Unbalanced Characters: Some characters in the game are considered to be ‘OP’ (overpowered). This can ultimately cause a lot of frustration for players that come against those characters and gives the user using these characters a significant advantage. Hence a lot of players complain it is no longer an even playing field and is unfair

- Microtransactions: While yes Brawlhalla is a free-to-play game, the game’s balance and gameplay can be hugely influenced by high expenditure from players. While this is the case for nearly all video games, Brawlhalla especially reaps a high reward due to the lack of brawler selection during promotional events. This brings me to my next point

- Brawlers during events: With the vast roster of characters in Brawlhalla, it makes sense that the game developers release/unvault some brawlers at a specific event or promotional collaborations. So hence this means unless you have purchased the brawler from the shop using in-game currency, you will lose the ability to use them until they are available in an event again. Hence this is where the spending comes in so players can secure their chosen brawlers. While yes it is possible to unlock and keep these brawlers without spending, it requires a long time on the game

- Is it really free to play? : the short answer is yes it is. However, from personal experience the game is very time-consuming and getting to the same level as someone who has spent money on the game takes a lot of time. This again topples the playing fields

Impressions:

From Brawlhalla we can take inspiration from its crisp graphics which I intend my game to have. Furthermore, Brawlhalla is free. However, I aim to take feedback on how Brawlhalla can be improved and not rely too much on in-game currency in my game. In addition, I would like to address the issue of character imbalance by making the individual character damages not too different from each other. Some attacks may cause higher damage, and some may be less. This allows the player to think more about his combinations of attacks and movement which overall improves hand-eye coordination and strategic skills

Street Fighter V

Street Fighter is arguably the most iconic fighting game franchise with its first game being made in 1987 by CapCom. Since then there have been 5 iterations of the game and I'd like to analyse the latest version which is Street Fighter V

Pros:

- Story Mode: Can appeal to narrative-driven individuals. Furthermore, a story mode is not present in most fighting games

- Deep Gameplay: Like Brawlhalla, Street Fighter V involves in-depth gameplay which is influenced by strategy and technique. The vast variety of moves, special moves and special

attacks for characters allows the player a wide range of playstyles and combinations which can appeal to some audiences

- Diverse Roster of Characters: Each character has different attack styles, strategies, moves etc. This allows players to find and explore characters which suit their own playstyle and encourages players to experiment.

- Microtransactions: Unlike Brawlhalla, Street Fighter V does not rely as heavily on microtransactions and players can progress much easier

- Multiplayer system: A highlight of this game is its matchmaking system. It is designed to ensure fair and balanced matches with players of similar calibre and skill levels. Furthermore, the game's net code is optimised for low latency and smooth play

Cons:

- No arcade mode: In my opinion, an arcade mode is synonymous with this franchise which originated in arcade/gaming centres. Ignoring it was a mistake from the game developers and this was quoted as "bizarre" by critics across the world

- Price: $60 for the game. Definitely on the expensive side for video games in general.

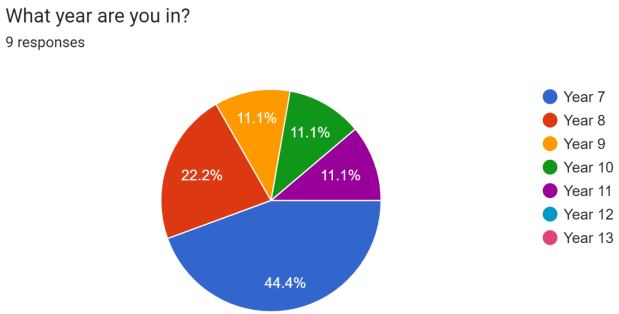
- Complexity of the game: Street Fighter V has received criticism for its combat system which is described to be incredibly deep and complex. While this may be a selling point for frequent buyers of the franchise, the learning curve to learn the game for new users is too steep. This may discourage new players to stick to the game as mastering this combat system takes a long time

Impressions:

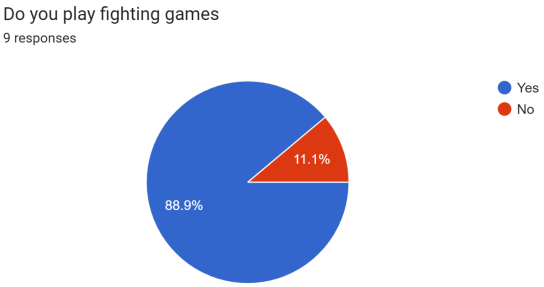
As we can see from street fighter it is an iconic video game series which has been around for a very long time. I chose to take inspiration from the game's lack of reliance on microtransactions and its diverse roster of characters. However, as stated above I aim to make my game free. Additionally, I aim to keep the control system of my characters simple and fun to ensure players still come to the gaming club.

Questionnaire

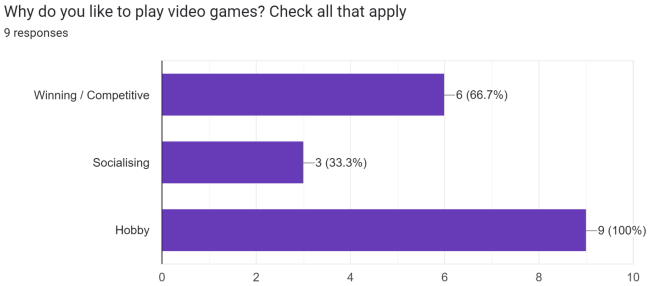
I have made a questionnaire which will be filled out by members of the club. Here are the results:



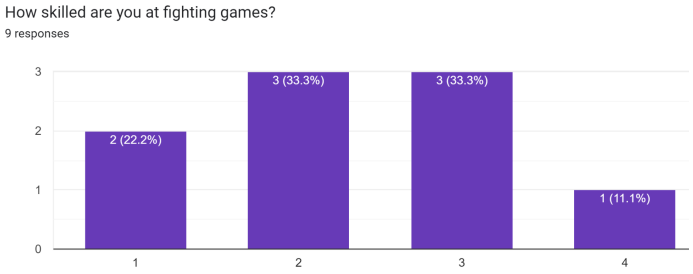
Here is a pie chart of the responses for the age group of the club. As we can see most of the target audience are Y7s followed by Y8s. This aligns with my aim of the game to make it for children hence the violence and features of the game shall be kept to PG 12



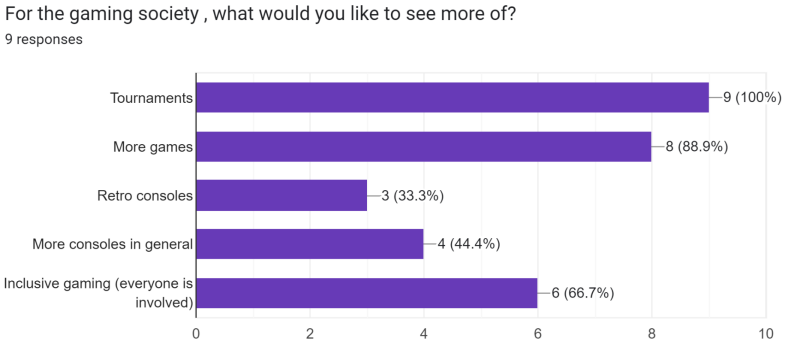
Here is a pie chart whether or not fighting games would be a good genre to introduce to the club. As you can see from the response ratio. Nearly everyone in the club has played a fighting game. This is good to know that the players know what to expect and what they should expect from my game



Here is a bar chart to show the main reason why people play video games. Most people play video games as a hobby and for the thrill of winning over actually socialising. This is perfect as my aim is to fix this socialising issue in the club and create this game to help people better socialise whilst also maintaining the competitiveness of the game



Here is a bar chart to show what everyone believes their skill level is in a fighting game genre. Everyone estimates their ability to be somewhat average with only one person on each of the Level 1 and Level 4 category. This suggests that most people are not confident in their abilities regarding fighting games. My game will have a practice mode to help the player practice and improve their skills against AI



And finally here is a chart to show what the members want from the club itself. Majority have voted for regular tournaments followed by everyone getting involved. This suggests I need to make my game player v player , have some sort of wireless connection which allows players to face off against each other. Or I could just make it so both players share a keyboard

Client Interview:

Here is an interview with my client, a regular member of the gaming club:

What is the general agenda of the gaming club every week?

There isn't really an agenda or a certain activity. Everyone just comes in and plays their choice of the game maybe with their friends however yes most of the time there isn't much going on. I just like to go there as I can play games in school whilst eating my fish and chips

This shows the lack of engagement and the enthusiasm by my client is rather low and most likely the same thing for the rest of the club. Overall boring environment

As a regular member would you say the club is popular?

No. Not many people want to spend their Friday lunchtimes playing games however I and my friends hang out there

shows us that the club is not exactly popular and it obviously stems from the lack of things to do

If there were enough members would you like to see tournaments? And why?

Tournaments would definitely be an interesting addition to the club. It will make lunchtime less boring and we can all compete with each other. And prizes would also be good along with our current gaming minutes currency

Tournaments are arguably the best thing to introduce to the club as it brings people together and makes the atmosphere more lively. However the main issue is why would the club members choose to play my game over other popular games? Let's find out

If you were to pick a game to compete against your friend. What would it be? Why?

Most likely FIFA or SmashBros. These games provide you a base skill level. What I mean by this is that there won't be any other ways in which the opponent player will have an advantage unless, of course, the player experience is different.

This tells me a competitive PvP game is what we need for this club. Additionally I believe that my game can be better than these big brand video games. First is that my game will not be reliant on microtransactions and 'overpowered' characters which makes it fair second the school need not invest in gaming consoles for a club that is barely popular.

What is your opinion on a game like smash bros except it is only exclusive to members of the club and everyone can play it?

No as I would much prefer to play my nintendo 3ds on my own

As stated previously this is the reason why the club is not as good as it used to be. Students simply go to this club as they are not allowed their consoles elsewhere in school .But my game of course will not require a console and in addition most schools have blocked websites for most games so my game, if good enough, can change my client's mind and be accessible anywhere in the school without blocks

Analysis of the interview

Analysing this client interview we can clearly see that the club itself is not too popular and not much activity goes on and that a competitive game would be highly beneficial. In terms of the question where it asks what type of games they would like to play against their opponent they say FIFA and Smashbros which are competitive games based on skill and timing. Which is what my game will be.

In addition I believe that user identification and accounts for every player will be helpful. This will help to organise tournaments and store wins for every player. In terms of the currency, I was told by my client this is something that was already established in the club as a running total currency so therefore I choose not to alter it too much

From what I have gathered from the client interviews and the questionnaire here is my success criteria

Program Specification

Menu Screen

- User is presented with a menu screen with a background image loaded up. A user interface is presented to the user with buttons for navigation: Play, Sign Up, Recent Fights, and Exit button

- The Recent Fights button will show users the winners of every round in the past fights that have occurred. This button will open a window to show this. Each round is displayed along with players involved in the fight. And finally, the winner of the round should also be displayed. This will help me to store round wins to their corresponding data

- New players will be able to create an account via sign up button. A user must create an account before playing the game or else they will not be able to play the game.

- To begin the game. Players will press the play button which should bring them onto the login page where they can enter their credentials and can view their stats

Sign up Page

- Should display fields for the new user to fill in
- Basic credentials must be taken such as username, password and email
- A check must be carried out to ensure that all fields are filled out
- An additional check must also be carried out to ensure the user's passwords match
- Yet another check should be carried out to ensure that the email address is not already registered
- Final check should be carried out to check the syntax of the email to see if it is valid
- If all fields are checked and unique. The new user's credentials should be stored in an external table. They are no longer a new user and these details should allow them to log in and play the game

Login Page

- After the play button is pressed. A login page should be presented for each of the players. Valid details will let them successfully log in and play the game
- In addition, a 'forgot password' option should be presented. Users can use it to reset their password via email

- In the event that a user chooses to reset their password. The user should be presented with an external window. This external window will allow users to enter their details and the corresponding email will receive a code which they can use to reset their password

- After a successful login, an option should be presented to each user where they can view their stats such as total playtime, total wins etc. This should be presented in an external window

- If both players are logged in and the start button is pressed. The Character Select screen should be displayed for them

Character Selection screen

- Both players should progress to a window to choose their character for fighting

- This will be a dropdown menu for each user. They can choose from a total of 5 different characters. Each character should have different damage properties.

- A preview of the character should be shown before each user confirms their character

- When both users have selected their character then the game can be started

The game itself

- The orientation/layout should begin with player 1's character on the left and player 2's on the right
- Health bars for each player should be displayed at the top along with the score counter below it
- Player characters should be able to move in any 2D direction and should always face the opponent character
- Each character should have different attacks. Which can be used against the opponent to deplete their health bar. The damage dealt should be designed to depend on the character itself and what attack they are using
- The first user wins a round by depleting the opponent's health bar
- The whole game should be 5 rounds in total. The player to win the most rounds will win the entire game

DOCUMENTED DESIGN

Overview:

A variety of components will be required for the main functionality/design of the program itself:

- Main Menu screen - Linking the components of my program together for my users
- The recent fights window - showcases all the fights that have happened in the game helping with point scoring and leaderboards
- Login page / signup page
- The main game itself

The main important part of my program is the game itself. I believe I need to make the game first and then build everything around my game such as the UI, statistics etc.

The rules of the game are as follows:

- Players play a total of 5 rounds
- First player to deplete the opponent's health bar will score a point. Players can do this through attacking the opponent based on what attack they choose to execute on the opponent (dependant of character chosen too)
- The player with the most points after 5 rounds is declared the winner

User Interfaces

Here is a very initial electronic visualisation at what the game itself presents to the user. I aim to keep it simple and effective so users can easily navigate it

As stated above this is an initial visualisation of my main menu screen. I feel this design of the interface screen for the main menu screen would be easy for the user to navigate and interact with because there are not too many icons/options cluttered into a small space hence allowing ease of access.

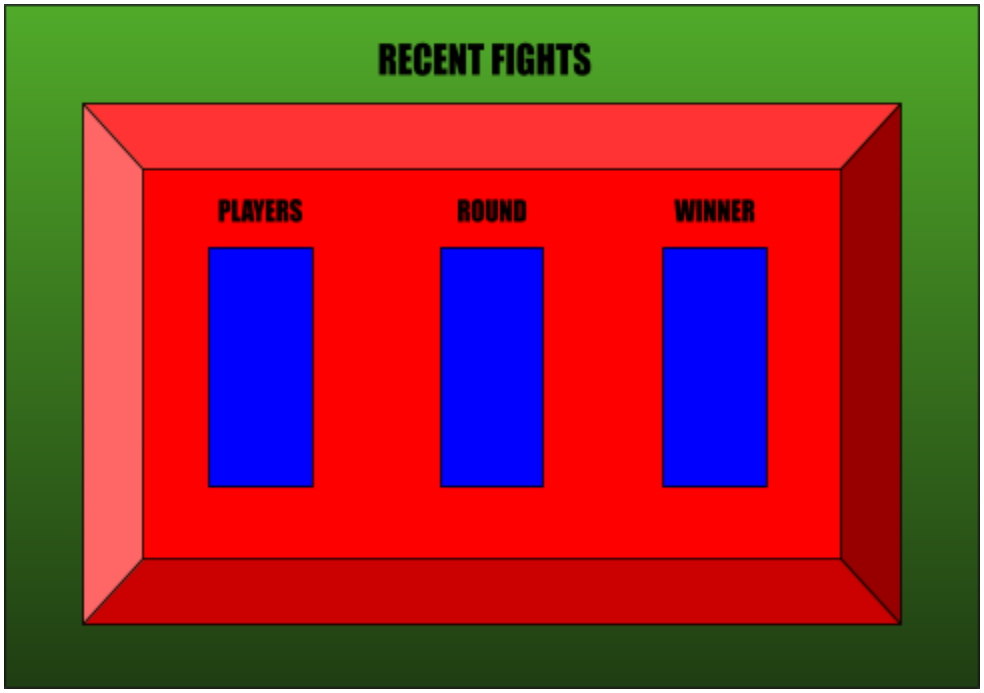
In addition to this, I feel like in my final product I should aim for a style of design would be eye catching and engaging. The **PLAY** button will entail a login screen which later brings users to the game in a local multiplayer format (2 players).

The **RECENT FIGHTS** area will contain the game histories which will help users to see past fights on the program.

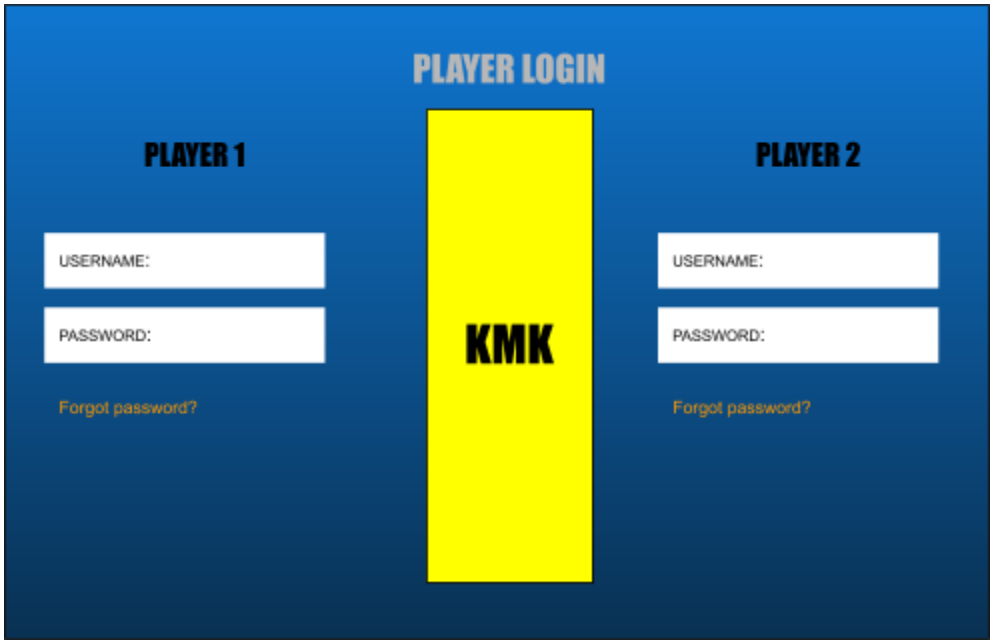
See below for the **SIGN UP** window

As stated in my analysis the game cannot be played without the users signing up. This window will allow the users to enter their details and their emails so they can login to play the game

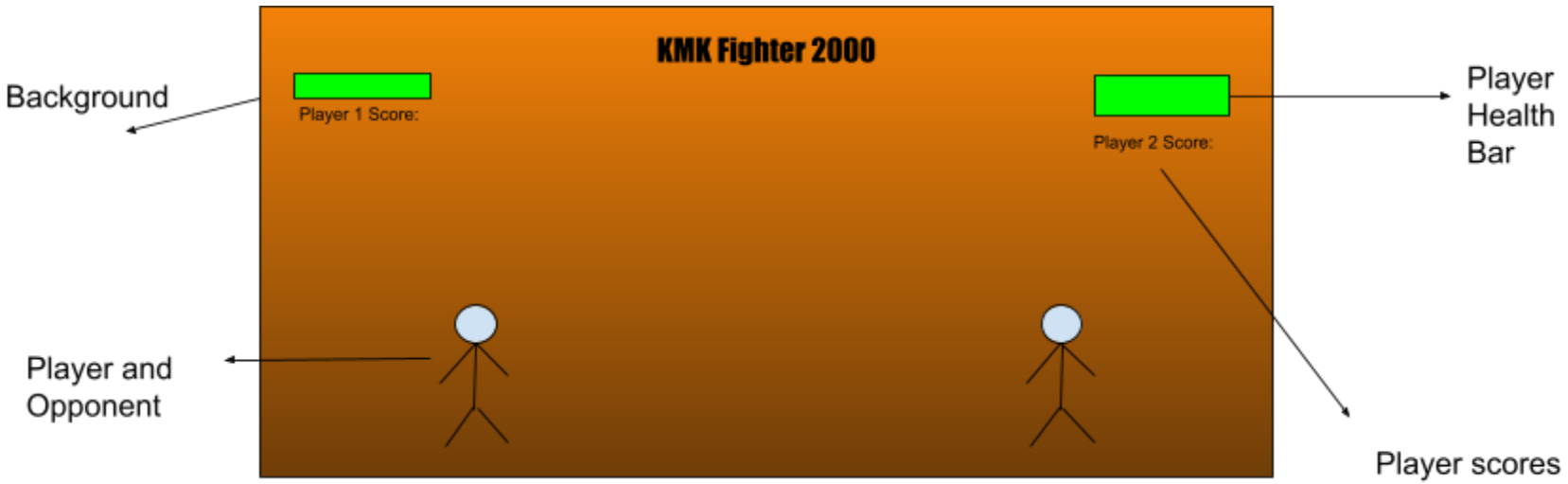
The fields themselves (shown later) will be a disappearing field. I feel this feature will be helpful in helping new users where exactly to enter their details



The recent fights window. The functionality of this window is to show the game histories and can also be used to quantify user wins for their stats (implemented later in the player stats SQL database). Usernames and winners will be synchronised and I aim to implement the player statistics into this screen as well. This will show the top players of the game



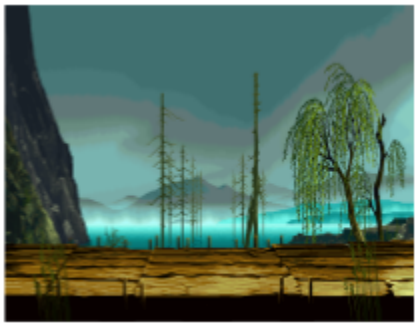
After clicking the PLAY button ,it will lead the user to this page. I will develop this page so that two players can log in by inputting their username and password simultaneously . This is login page/details is essential for the player statistics system because users can store their performance over periods of time, hence leading to a more competitive and rewarding game for players.



This illustration represents two players playing a normal game. There is a player score displayed below the health bar. The health bar will be designed to deplete as an attack on the opponent is executed. As stated before the score will be incremented and the most wins out of 5 rounds will win the game This will all be developed in Pygame.

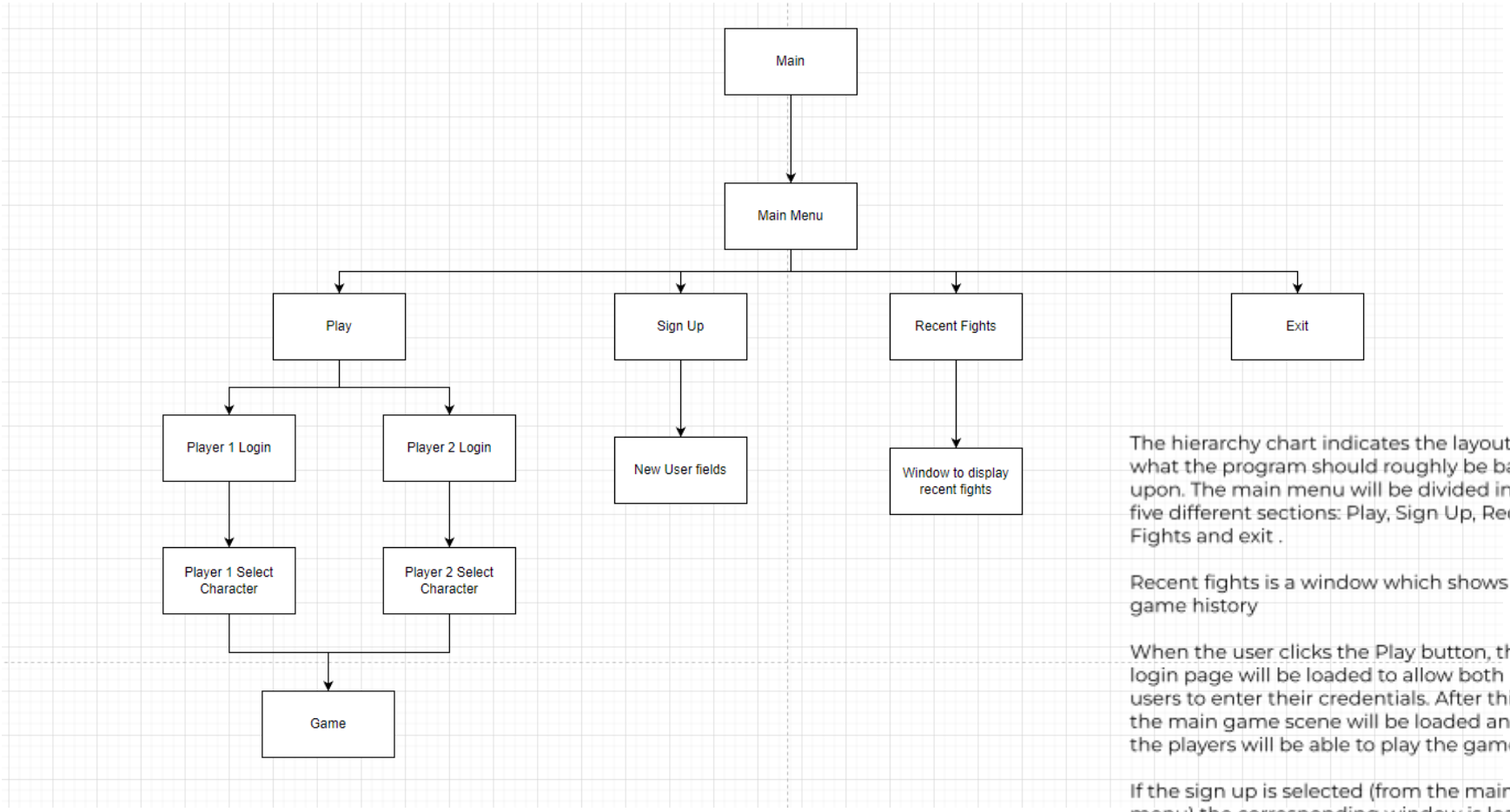
Assets of the game

Backgrounds



Hierarchy chart

shows the different stages of the game



The hierarchy chart indicates the layout of what the program should roughly be based upon. The main menu will be divided into five different sections: Play, Sign Up, Recent Fights and exit .

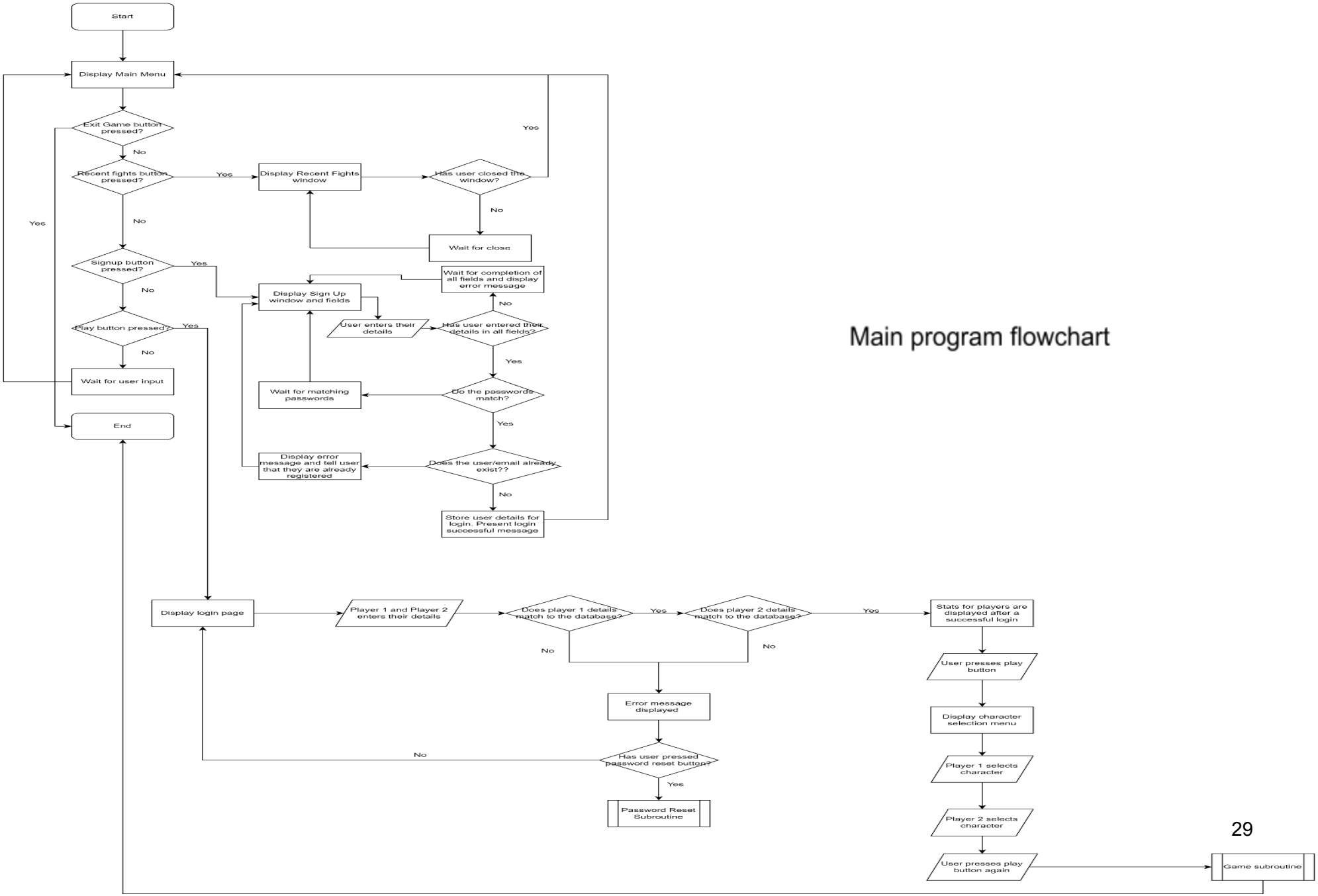
Recent fights is a window which shows the game history

When the user clicks the Play button, the login page will be loaded to allow both users to enter their credentials. After this the main game scene will be loaded and the players will be able to play the game.

If the sign up is selected (from the main menu) the corresponding window is loaded for the user to use. This is where the user should be able to register.

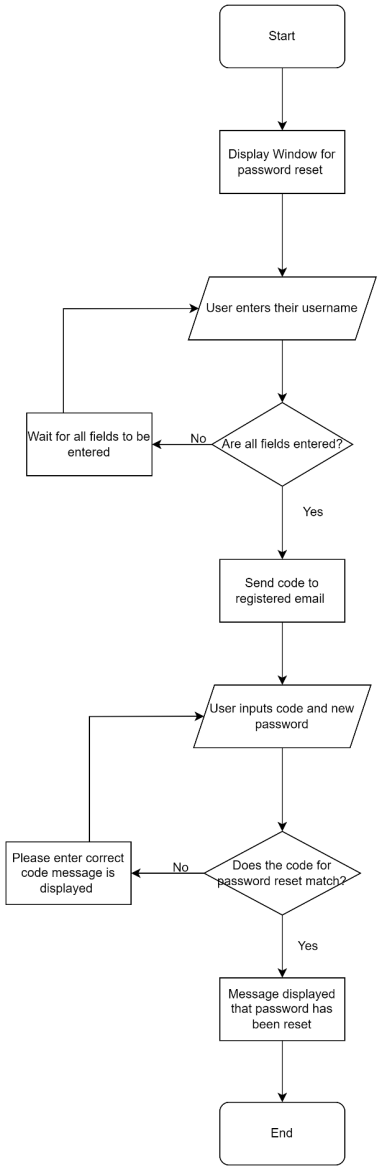
Flowcharts

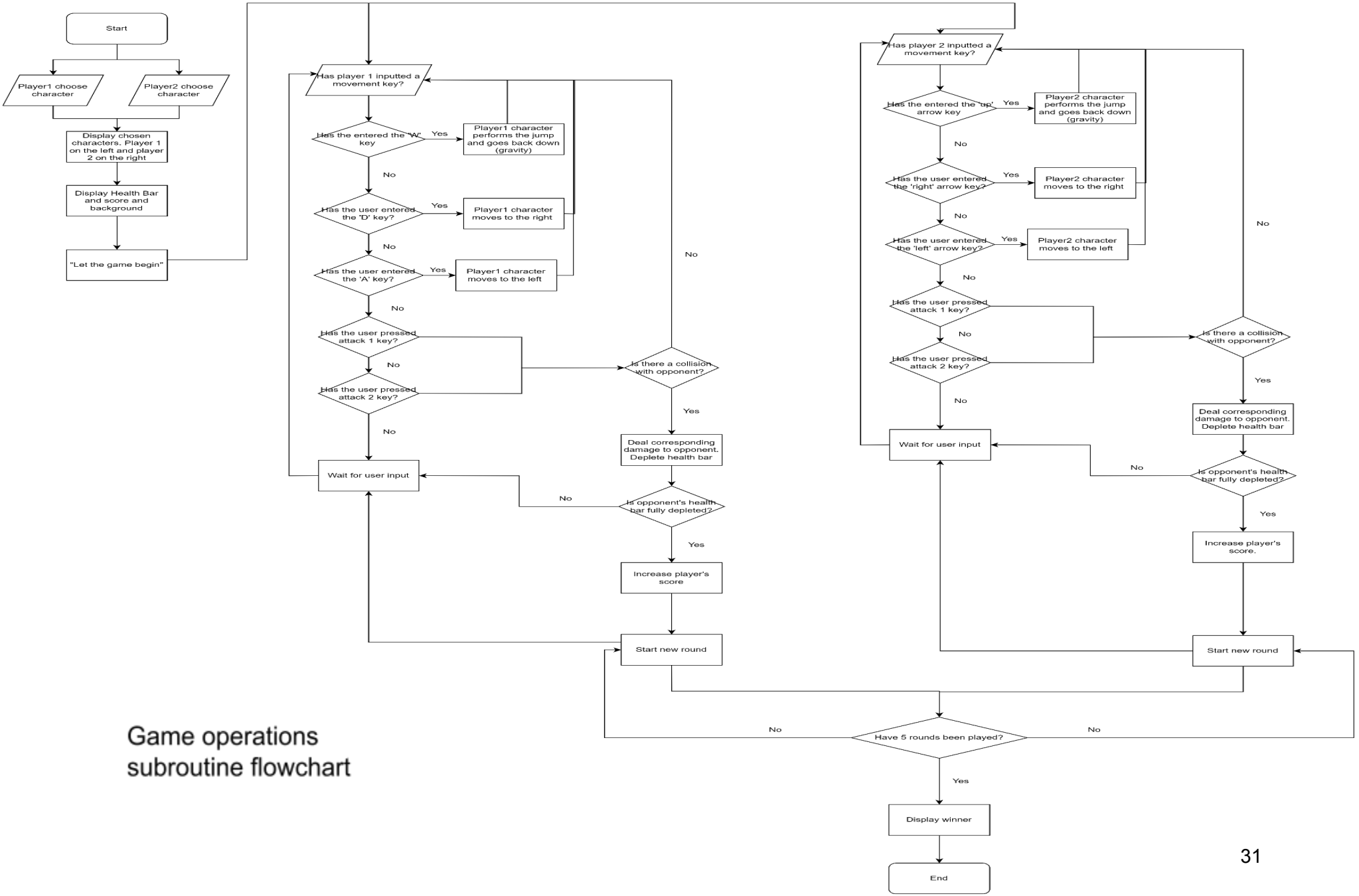
shows the stages of the game as a user navigates through it. There are 3 main ones in my program



Main program flowchart

Password reset subroutine flowchart





Game operations subroutine flowchart

Implementation of Important Algorithms pt 1

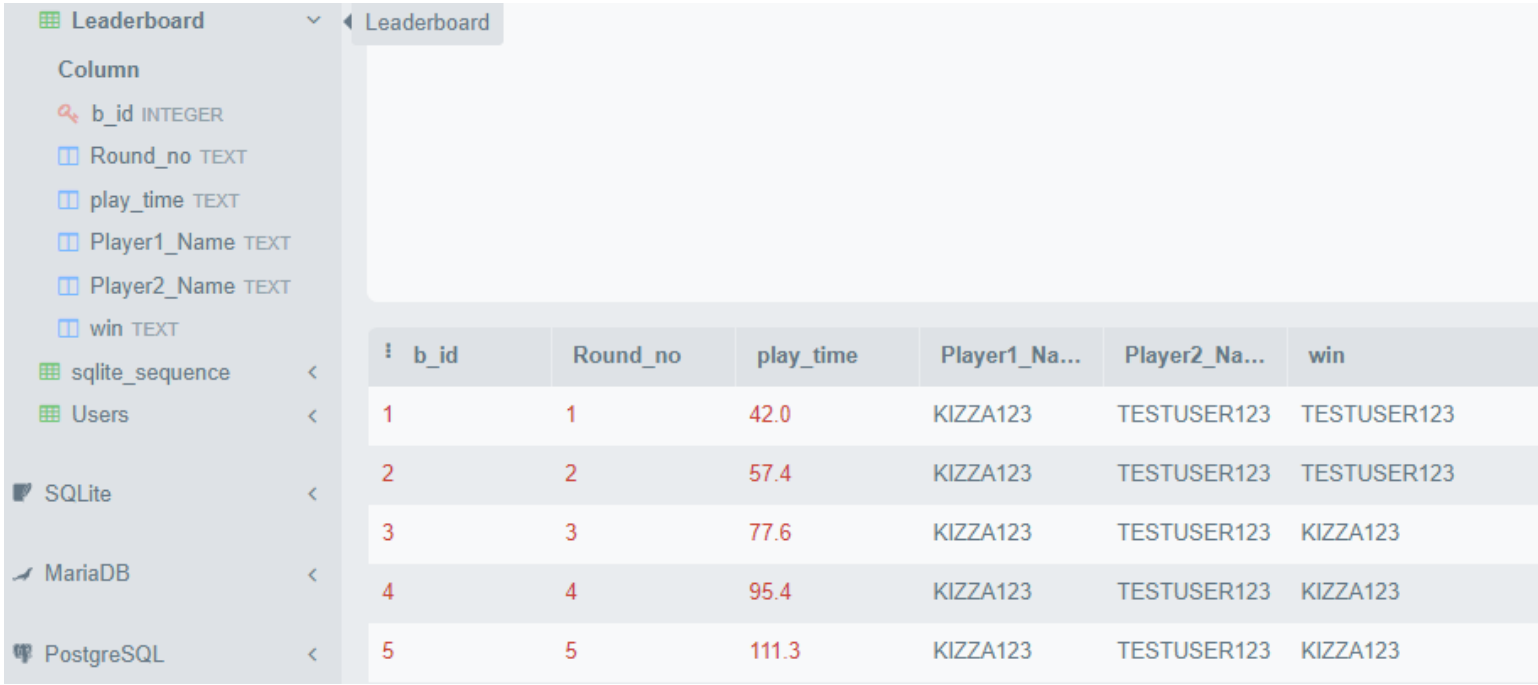
As stated above, I have been tasked to store the users who have played the game. I decided that multiple linked SQL Tables is the best way to carry out this task

Total of 3 separate tables each intended for a different function:

User table : Where all user credentials of the game are registered to and stored
Recent Fights : To show all of the fights that have occurred in the game
Totals: Things like total user's and total rounds to help us with aggregate SQL functions

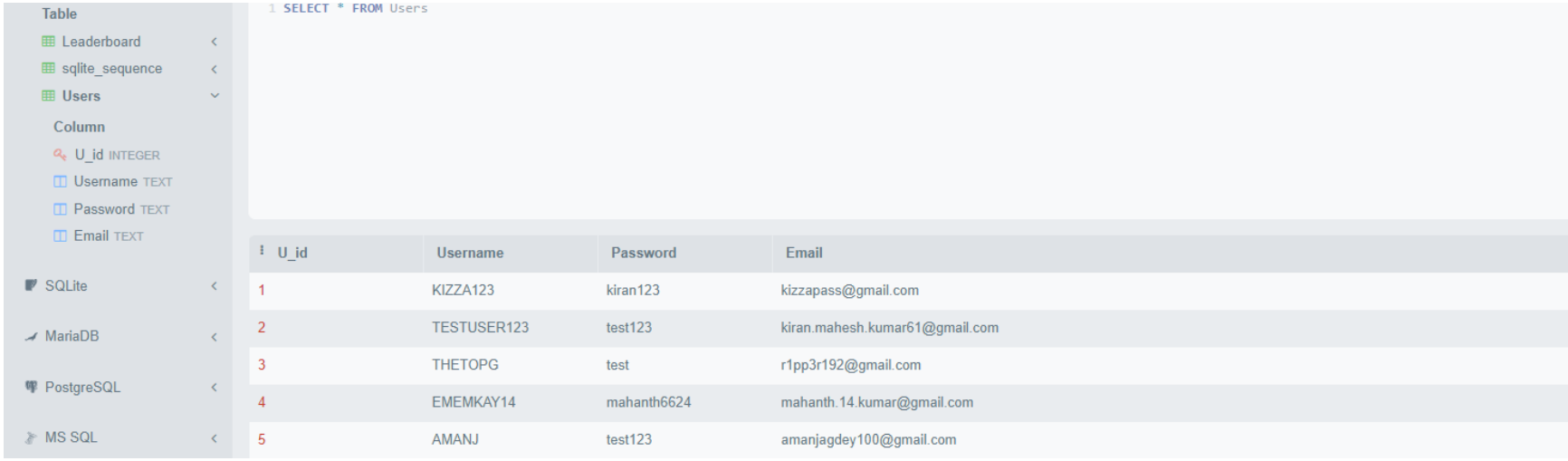
Each of these tables will be created using SQLite library and connected SQL Online IDE

Here is a look at the structure of the tables:



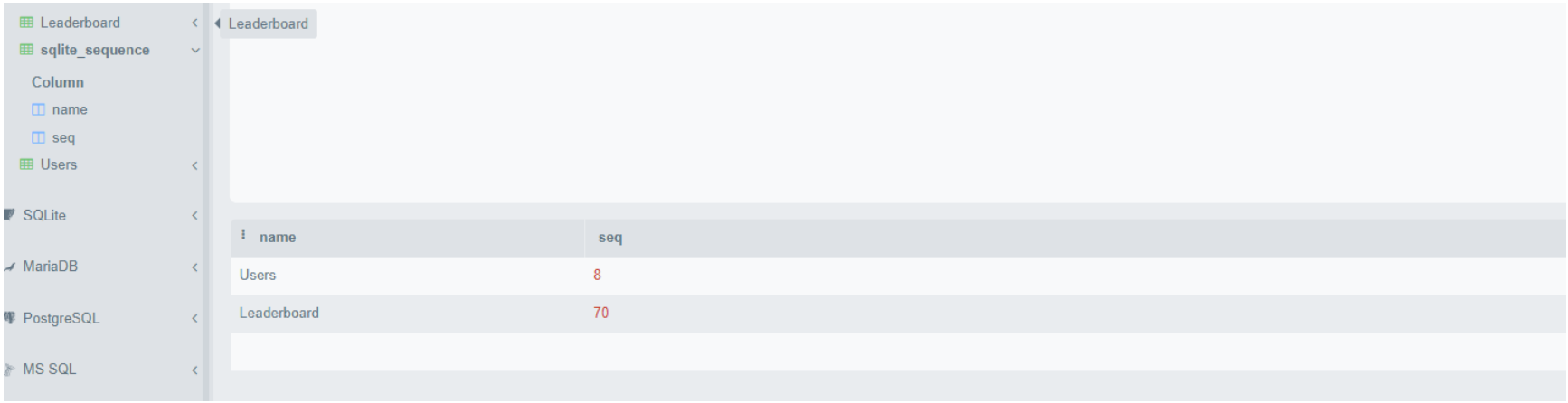
b_id	Round_no	play_time	Player1_Na...	Player2_Na...	win
1	1	42.0	KIZZA123	TESTUSER123	TESTUSER123
2	2	57.4	KIZZA123	TESTUSER123	TESTUSER123
3	3	77.6	KIZZA123	TESTUSER123	KIZZA123
4	4	95.4	KIZZA123	TESTUSER123	KIZZA123
5	5	111.3	KIZZA123	TESTUSER123	KIZZA123

Above is the recent fights table. This table consists of 6 attributes. Battle ID is the primary key and this is used to link the user tables. Round_no represents the round number and the playtime is calculated for the play_time attribute. This is later used in calculating the average playtimes etc. Furthermore the table displays the 2 players and the winner of the round. This is the main table used for individual player Stats



U_id	Username	Password	Email
1	KIZZA123	kiran123	kizzapass@gmail.com
2	TESTUSER123	test123	kiran.mahesh.kumar61@gmail.com
3	THETOPG	test	r1pp3r192@gmail.com
4	EMEMKAY14	mahanth6624	mahanth.14.kumar@gmail.com
5	AMANJ	test123	amanjagdey100@gmail.com

This is the users table. This is where the user's credentials are stored. Each user must login using their credentials that initially are from this table. Users can also be created and they are then stored onto this table. User ID primary key used to interlink tables and all calculations are stored under each user and displayed upon request



And finally this table shows us the total number of users and the total number of rounds that have been played in the game in its entirety. As stated above this is the main table used for calculations

All code and functions for my SQL tables are further explained in my technical solution

Below represents a data connection algorithm. This is used in the program for storing user data, calculating player stats like averages and for showcasing the proficient players of the game. Here it is in pseudo code format / plain english

Import sqlite3 library as sql

Define a DB class with the following methods:
a. init method which creates a connection to the database file, initialises a cursor, and calls a method to create database structure
b. method which creates tables for Users and Leaderboard if they don't exist
c. method which retrieves user information by their username
d. method which retrieves information about leaderboard
e. method which retrieves statistics for a given player
f. method which inserts data into the Leaderboard table
g.method which updates user's password by their username
h. method which inserts user data into the Users table

execute a select query on the table of users to retrieve user data by username, and return the result
execute a select query on the Leaderboard table to retrieve leaderboard data, loop through the result, format the data as a string, and return it

execute two select queries on the Leaderboard table to retrieve the total play time, average play time, and total number of rounds via Aggregate SQL functions played by a given player, return information

execute an insert query on the Leaderboard table to insert new data, and return a success message or an error message if an exception is caught

execute an update query on the table of users to update a user's password, and return a success message or an error message if an exception is caught

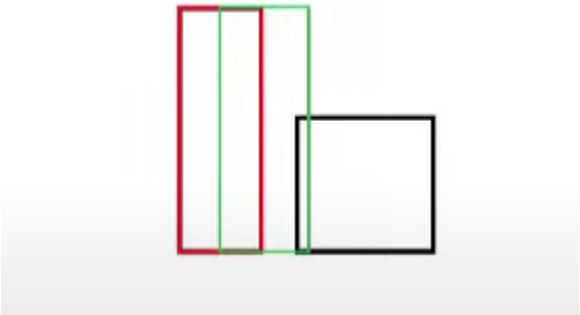
execute an insert query on the table to insert new user data, and return a success message or an error message if an exception is caught

Implementation of Important Algorithms pt 2

Collisions

Here is an image of what is going on when a player attack collides with the opponent. Let the red box represent the player's attack and the black box is the opponent. When the player attacks then he goes slightly into the block hence overlapping it. This is obviously not ideal and this happens because the code itself will not flag up the collision until the red box has touched the black box and by that time it has touched the box it is too late to flag up the collision and for the code to react. If the attack animation plays and THEN check for collision it simply won't work

Here is a solution:



In this case the green rectangle is saying that I have pressed the right arrow key so my red rectangle is intending to move to the right.

That’s the job of the green rectangle is to represent the player movement and it's a preemptive way of checking for collision before the actual player gets there.

So therefore if a collision is detected then we can’t move the player and then this allows us to code in restrictions for the player movement

Here is a pseudocode for how I intend to carry out this:

```
animation_cooldown = 50
if self.alive is FALSE then
self.frame_index = len(self.animation_list[self.action]) - 1
Else then
self.frame_index = 0
# check if an attack was executed
if self.action == 3 or self.action == 4 then
self.attacking = False
self.attack_cooldown = 20
# check if damage was taken
if self.action == 5:
self.hit = False
# stops attack
self.attacking = False
self.attack_cooldown = 20
DEFINE FUNCTION attack:
 if self.attack_cooldown == 0:
# execute attack
self.attacking = True
self.attack_sound.play()
# below is the execution for a collision
attacking_rect = pygame.Rect(self.rect.centerx - (2 * self.rect.width * self.flip), self.rect.y,
                             2 * self.rect.width, self.rect.height)
```

```
IF attacking_rect.colliderect(target.rect) then
target.health -= self.attack_damage
target.hit = True

DEFINE FUNCTION update action
# check if the new action is different to the previous one
IF new_action != self.action then
self.action = new_action
# update animation
self.frame_index = 0
self.update_time = pygame.time.get_ticks()
```

Implementing Important Algorithms pt 3

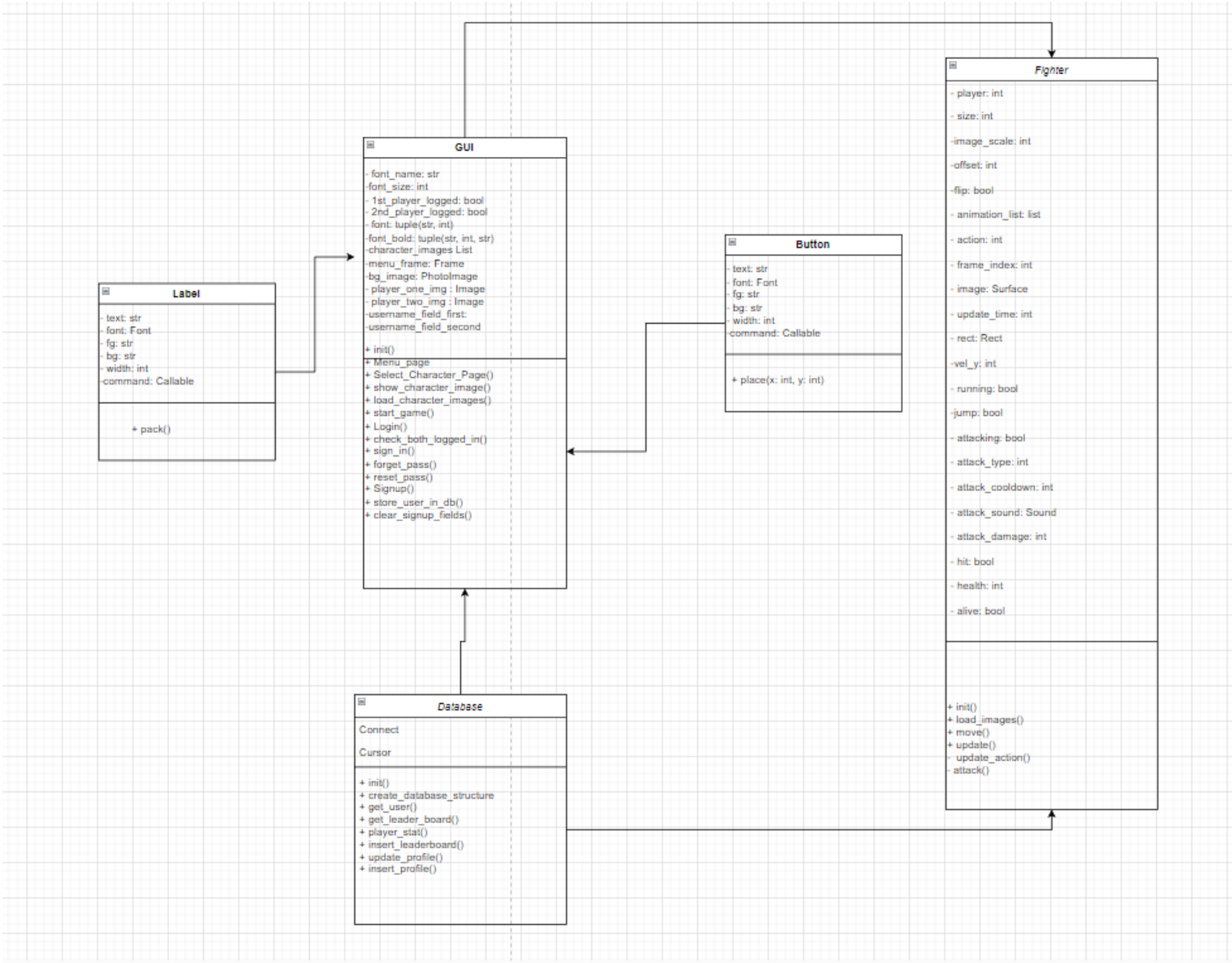
Animations are done by using individual images of a character from a spritesheet. This is implemented via a list operation. Essentially you have to iterate through them at a set frame rate and if I do it correctly and quick enough, an animation will be formed.

The spritesheet (depending on the character) will have all their actions in one big sheet and essentially I extract each “frame” / individual image separately and store them all within my python file

Here is a simple english / pseudocode algorithm for the animations and how it is carried out:

SET dimensions : 162
LOAD character.jpg
CHR_FRAME = [10, 8, 1, 7, 7] # this is the number of frames each category of animation has for this character it is 10 frames for the idle animation
# PASS INTO FIGHTER CLASS #
fighter = Fighter(200,310, character.jpg , CHR_FRAME)
CREATE METHOD IN FIGHTER
TAKE character.jpg , CHR_Frame
# EXTRACTING#
FOR EVERY ANIMATION then
EXTRACT TEMP IMAGE by dimensions
ITERATE BY CHR_FRAME based on CONTROLS

CLASS DIAGRAM:


Label
- text: str
- font: Font
- fg: str
- bg: str
- width: int
-command: Callable
+ pack()
GUI
- font_name: str
-font_size: int
- 1st_player_logged: bool
- 2nd_player_logged: bool
- font: tuple(str, int)
-font_bold: tuple(str, int, str)
-character_images List
-menu_frame: Frame
-bg_image: PhotoImage
- player_one_img : Image
- player_two_img : Image
-username_field_first:
-username_field_second
+ init()
+ Menu_page
+ Select_Character_Page()
+ show_character_image()
+ load_character_images()
+ start_game()
+ Login()
+ check_both_logged_in()
+ sign_in()
+ forget_pass()
+ reset_pass()
+ Signup()
+ store_user_in_db()
+ clear_signup_fields()
Button
- text: str
- font: Font
- fg: str
- bg: str
- width: int
-command: Callable
+ place(x: int, y: int)
Fighter
- player: int
- size: int
-image_scale: int
-offset: int
-flip: bool
- animation_list: list
- action: int
- frame_index: int
- image: Surface
- update_time: int
- rect: Rect
-vel_y: int
- running: bool
-jump: bool
- attacking: bool
- attack_type: int
- attack_cooldown: int
- attack_sound: Sound
- attack_damage: int
- hit: bool
- health: int
- alive: bool
+ init()
+ load_images()
+ move()
+ update()
- update_action()
- attack()
Database
Connect
Cursor
+ init()
+ create_database_structure
+ get_user()
+ get_leader_board()
+ player_stat()
+ insert_leaderboard()
+ update_profile()
+ insert_profile()

Breakdown of important classes on this diagram

DB class:

The DB class has a dependency on the sqlite3 module for creating and manipulating the database. It also uses the Exception class for error handling. The DB class has a constructor that creates or connects to the database and initialises a cursor for DML operations. The create_database_structure() function creates the tables if they do not already exist. The get_user(), get_leader_board(), player_stat(), insert_leaderboard(), update_profile(), and insert_profile() functions perform various operations on the database, such as selecting data, inserting data, and updating data. All of these functions interact with the database through the cur cursor object.

GUI Class:

The GUI class inherits from Tk and has several instance variables: font_name, font_size, _1st_player_logged, _2nd_player_logged, font, font_bold, character_images, menu_frame, and bg_image. It has an __init__() method that initializes the instance variables, sets up the GUI window, and calls various methods to set up the GUI pages and images.

The Menu_Page() method is a public method of the GUI class that handles the menu page of the GUI. It creates a menu_frame and sets its background image to bg_image. It also creates several buttons for playing the game, signing up, viewing the recent fights, and exiting the application. These buttons have various commands that either switch to a different GUI page, show a messagebox, or exit the application. This class inherits attributes from the in built Button and Label classes

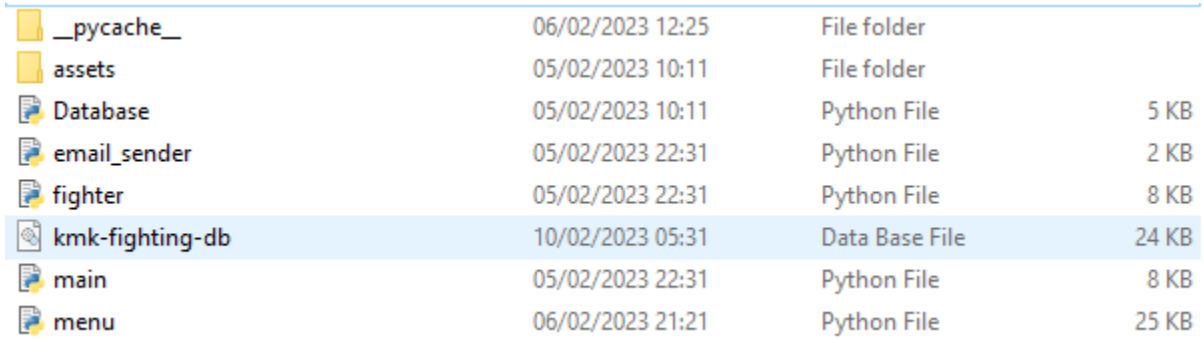
Fighter Class:

The Fighter class has instance variables for various properties of a fighter, such as player, size, image_scale, flip, animation_list, action, frame_index, image, update_time, rect, vel_y, running, jump, attacking, attack_type, attack_cooldown, attack_sound, attack_damage, hit, health, and alive. These variables is the blueprint for a character in my game

The class has an __init__() method that initializes the instance variables, and a load_images() method that loads images for the fighter's animations. There is also a move() method that handles player movement and keyboard input, and an update() method that updates the fighter's animation and status. The update_action() method is used by the update() method to set the fighter's current action, and the attack() method is used to handle attacks.

File and Asset Organization

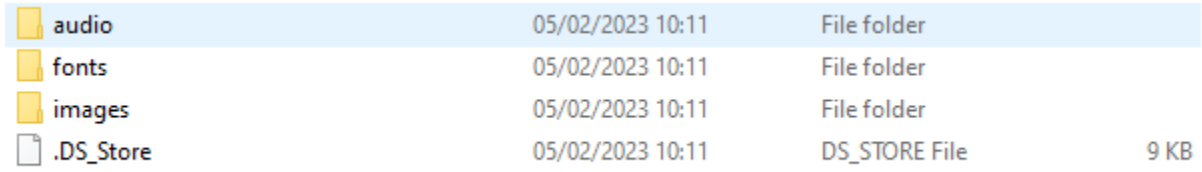
Main game folder



__pycache__	06/02/2023 12:25	File folder	
assets	05/02/2023 10:11	File folder	
Database	05/02/2023 10:11	Python File	5 KB
email_sender	05/02/2023 22:31	Python File	2 KB
fighter	05/02/2023 22:31	Python File	8 KB
kmk-fighting-db	10/02/2023 05:31	Data Base File	24 KB
main	05/02/2023 22:31	Python File	8 KB
menu	06/02/2023 21:21	Python File	25 KB

It contains the 5 scripts for all the functionality of the game. In addition the database file (kmk-fighting-db) can be opened externally in SQLOnline. All the python scripts use the assets folder for all the game's elements such as characters, login screen etc.

Assets



audio	05/02/2023 10:11	File folder	
fonts	05/02/2023 10:11	File folder	
images	05/02/2023 10:11	File folder	
.DS_Store	05/02/2023 10:11	DS_STORE File	9 KB

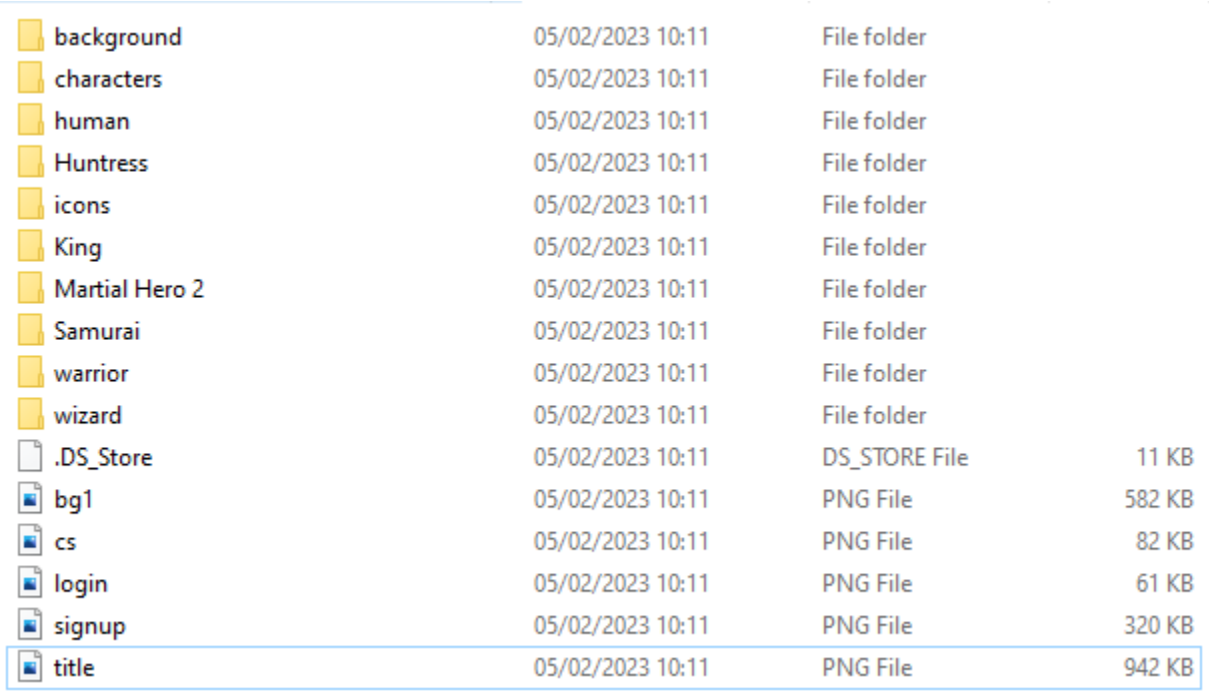
Containing the assets that are imported into the game.

The audio file consists of the attack sounds and the main game sound. In the code it is imported using a mixer module

Font File

This is the font file. Currently only consists of this font. This font is imported into the game for the player scores, countdown and the victory image

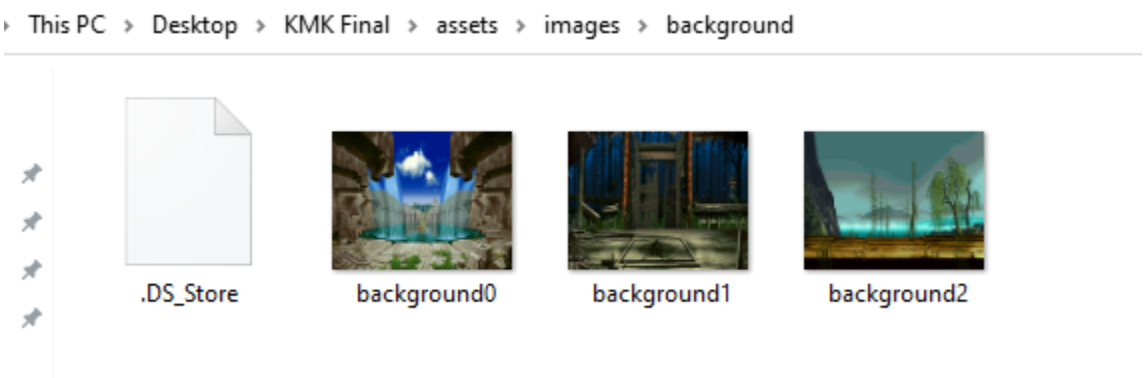
Images File



background	05/02/2023 10:11	File folder	
characters	05/02/2023 10:11	File folder	
human	05/02/2023 10:11	File folder	
Huntress	05/02/2023 10:11	File folder	
icons	05/02/2023 10:11	File folder	
King	05/02/2023 10:11	File folder	
Martial Hero 2	05/02/2023 10:11	File folder	
Samurai	05/02/2023 10:11	File folder	
warrior	05/02/2023 10:11	File folder	
wizard	05/02/2023 10:11	File folder	
.DS_Store	05/02/2023 10:11	DS_STORE File	11 KB
bg1	05/02/2023 10:11	PNG File	582 KB
cs	05/02/2023 10:11	PNG File	82 KB
login	05/02/2023 10:11	PNG File	61 KB
signup	05/02/2023 10:11	PNG File	320 KB
title	05/02/2023 10:11	PNG File	942 KB

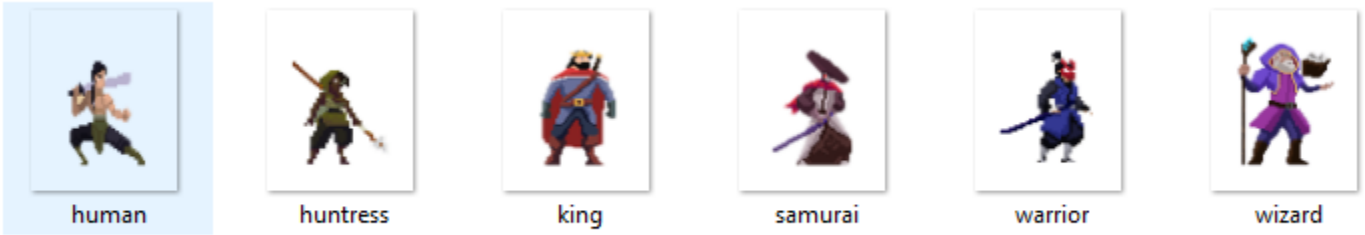
This is the main place of the assets for the game. Bg1 is the main menu background, cs is the character select screen. Login is the image used for the login page and likewise for signup

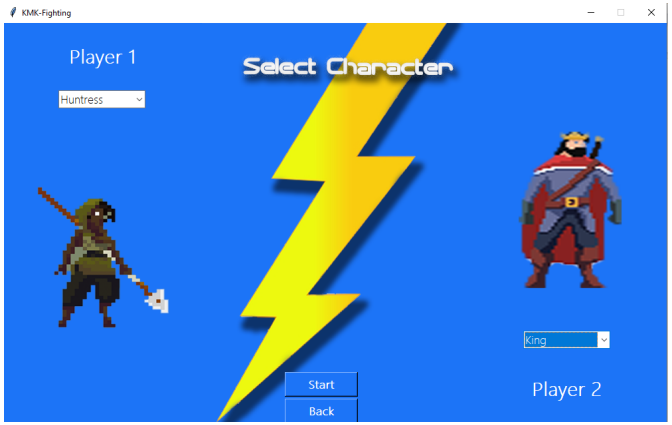
Background File



This is the background folder. These background files are imported to the main game when the game starts. The game randomises the backgrounds and blits the background. The background that gets displayed is total random

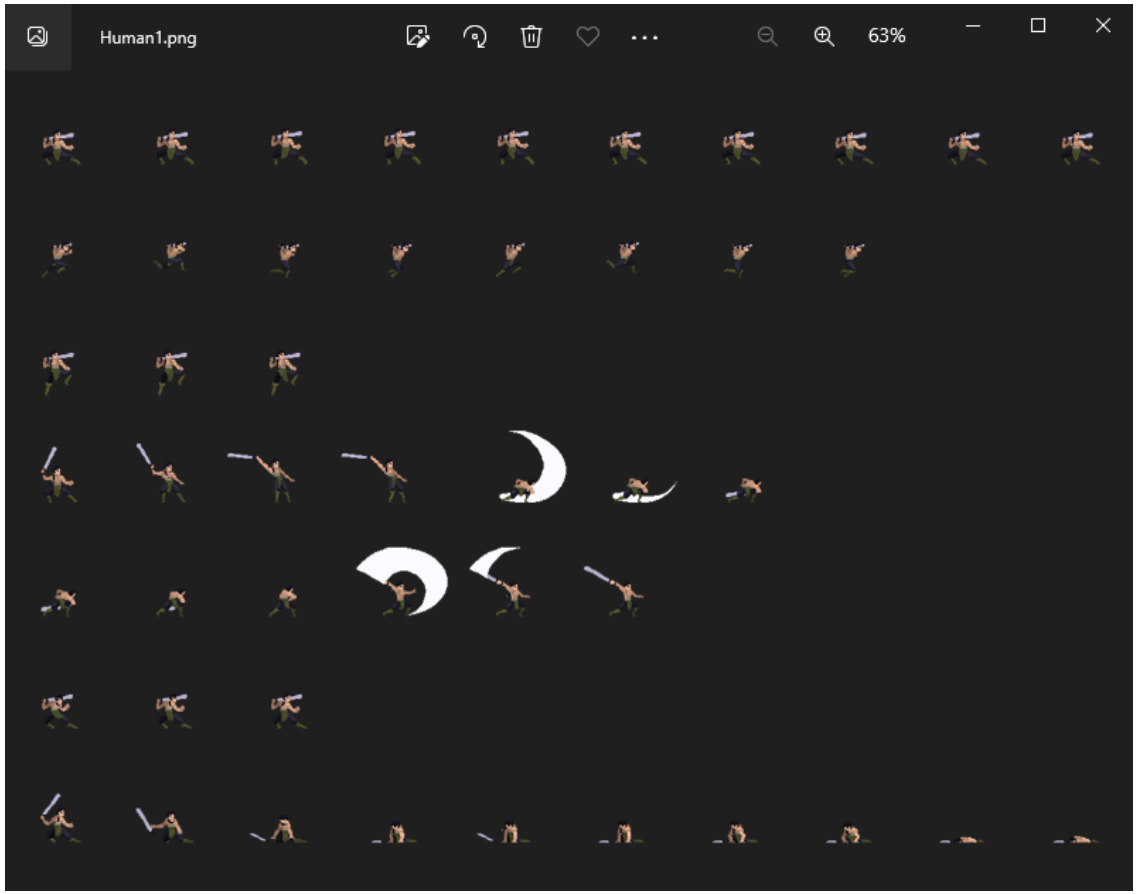
Character Select Icons (characters)





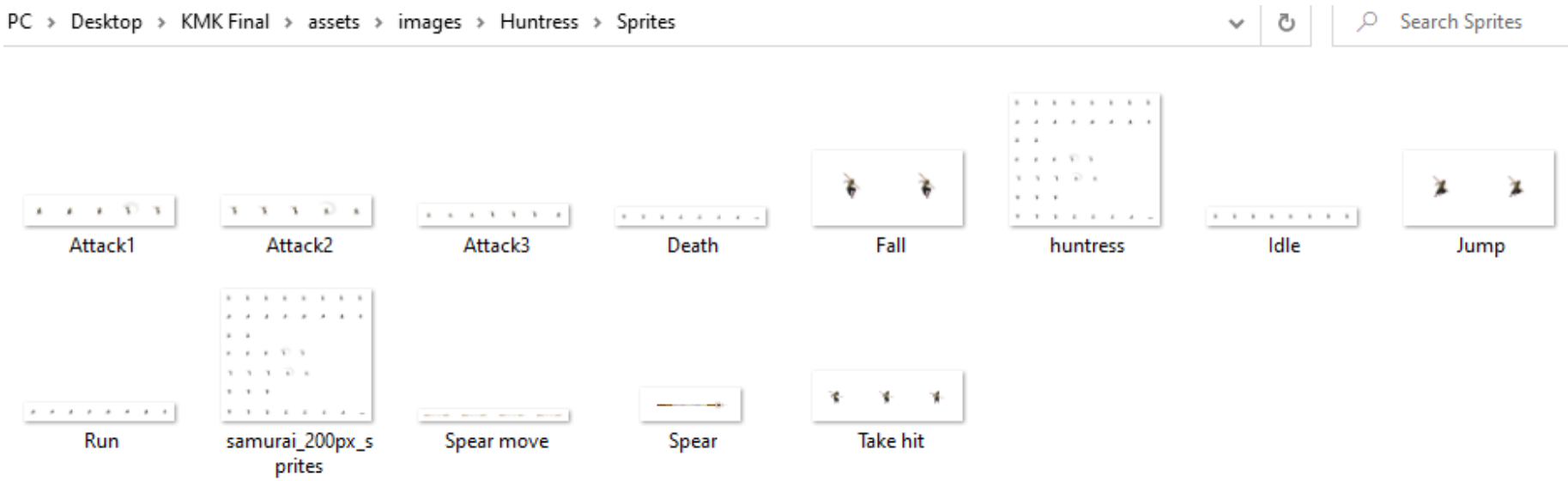
The characters folder is actually icons that are displayed on the character. This is used in the character select screen for the user to preview the character they want to choose

Sprite Sheets (Human)



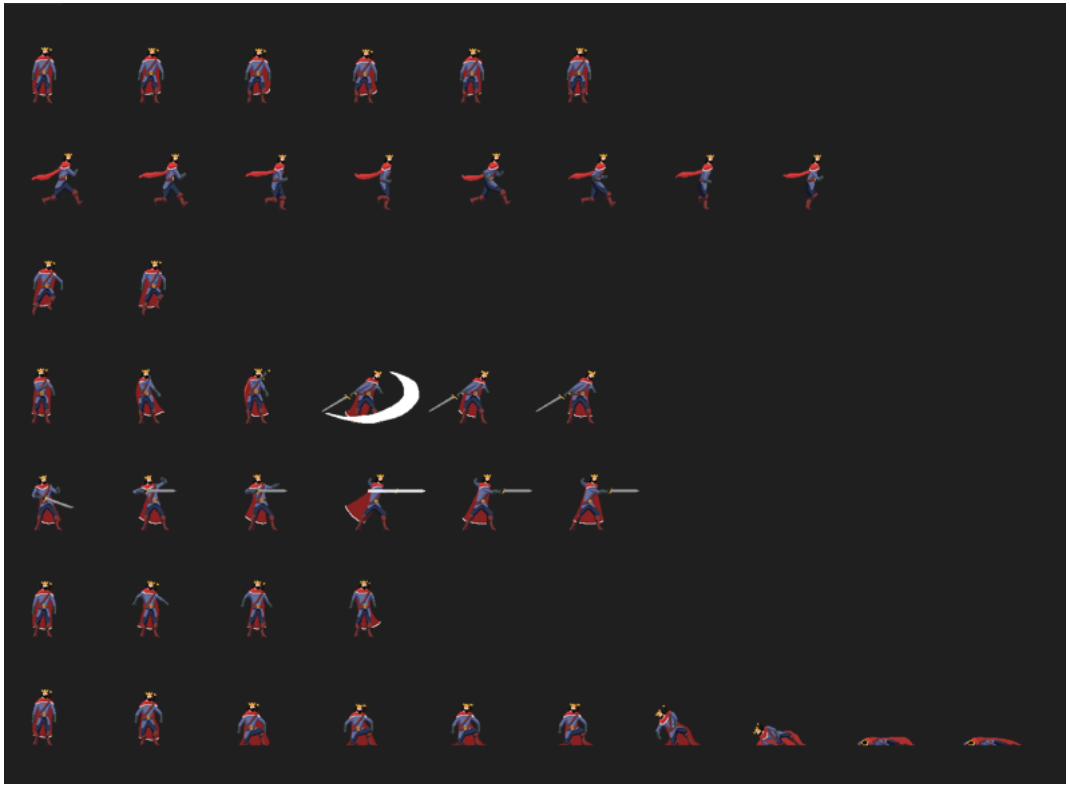
In the corresponding character folders, you can see from these images the characters are not actually a singular image with a singular attack. These are sprite sheets which correspond to an action by the player

Sprite Sheets (Huntress)

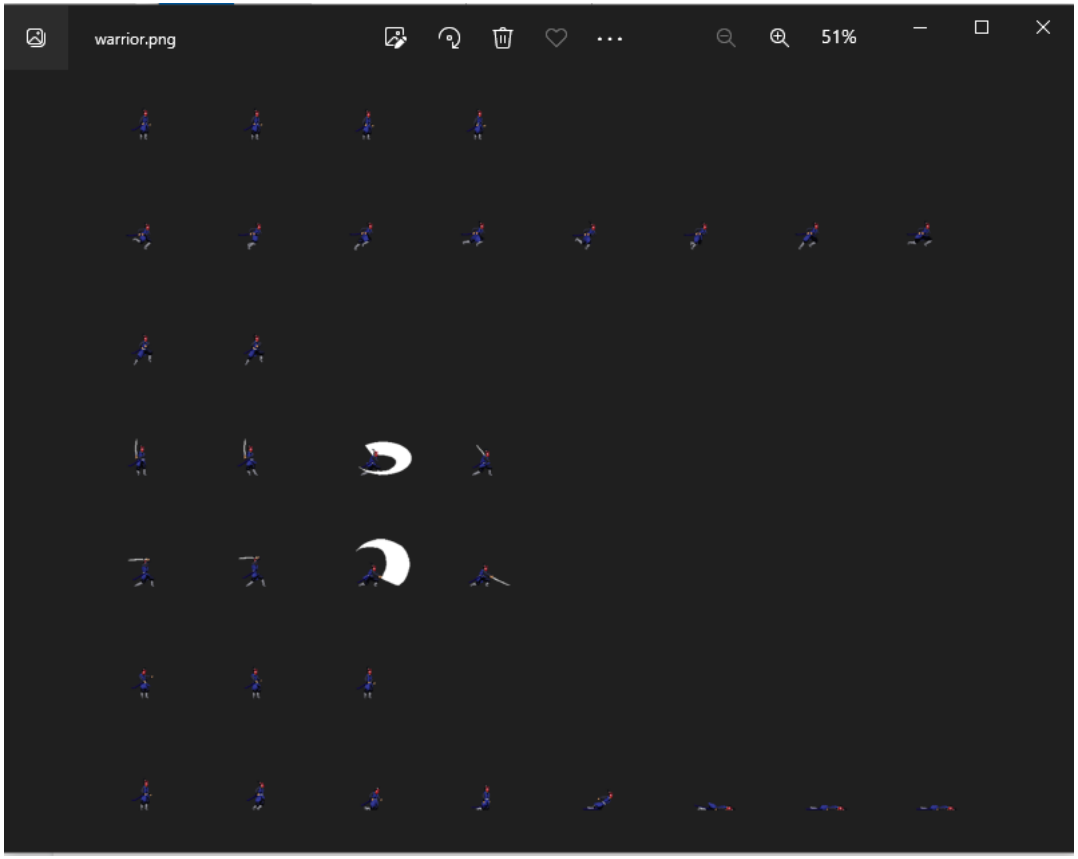


I chose not to group huntress as there were too many images to fit onto one big sprite sheet

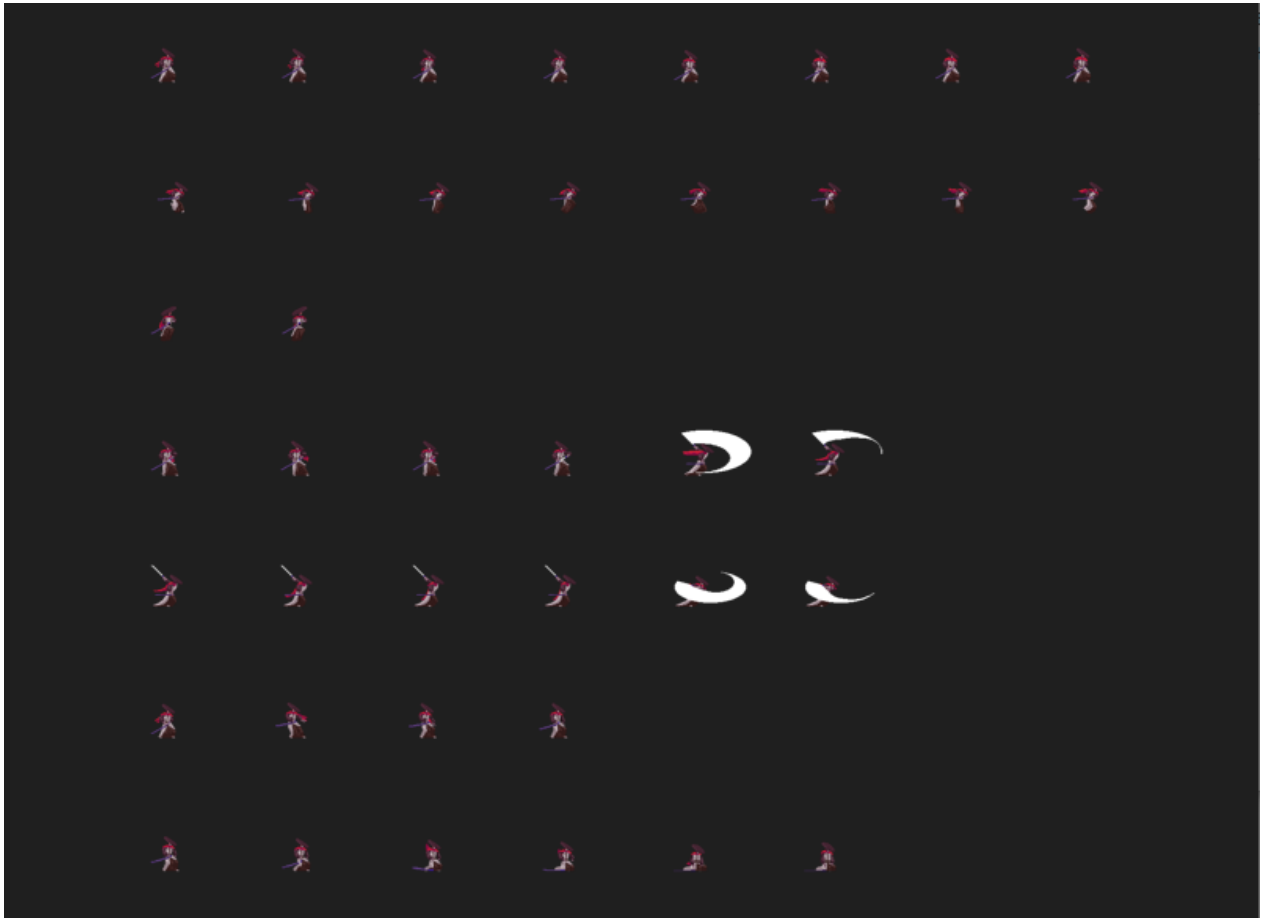
Sprite Sheets (King)



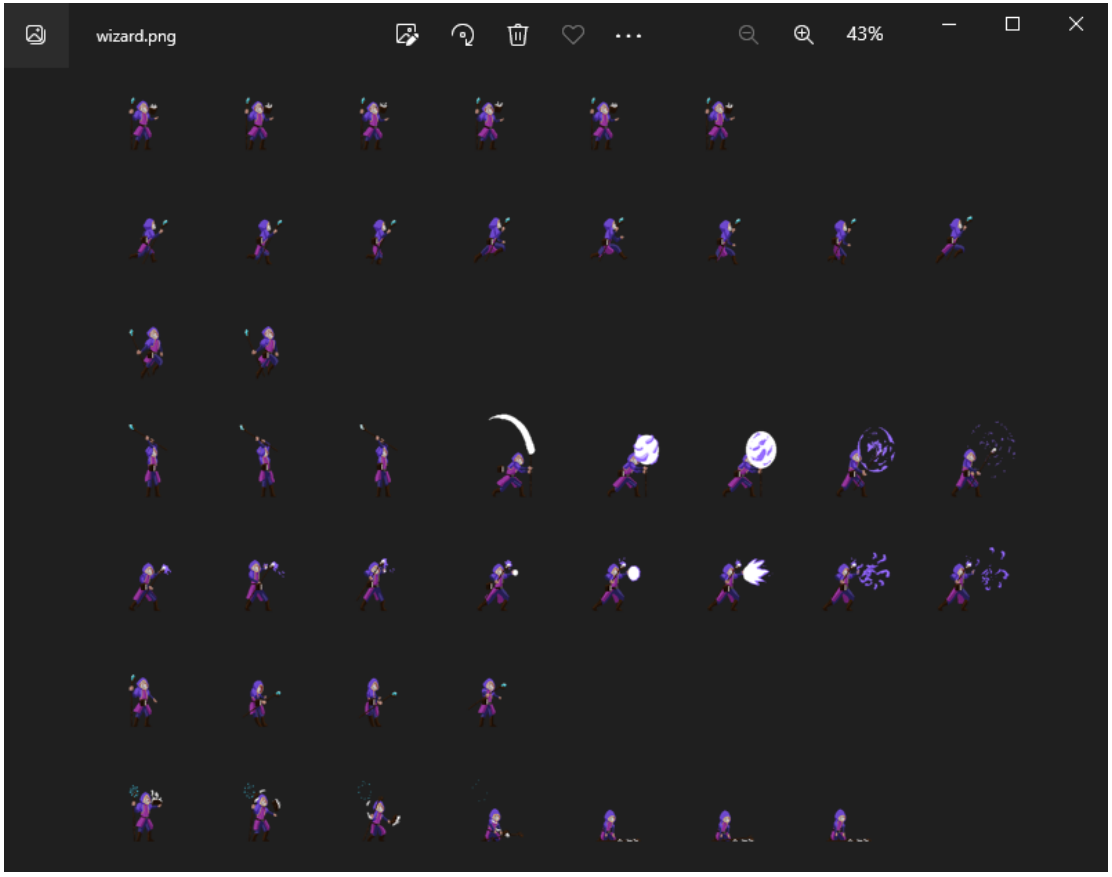
Sprite Sheets (Warrior)



Sprite Sheets (Samurai)



Sprite Sheets (Wizard)



TECHNICAL SOLUTION

```
                                    ##menu.py SCRIPT 1##
import sys, os, random, re
from tkinter import *
from tkinter import ttk, messagebox #important modules are imported such as tkinter for graphics and
other classes#
from main import run_fighter_game # main game function
from Database import DB
from email_sender import Send_email

db = DB() # database class


##validating/checking email function##
def email_validation(mail):
    pat = r'\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,7}\b' #syntax such as letters and numbers
are checked#
    if re.match(pat, mail):
        return True
    else:
        return False
##Email is validated via checking syntax of user input##
```

```
def widget_enter(event, widget, text): # this function makes it so the text to guide the user when
entering their details disappears
    if widget.get() == text:
        widget.delete(0, END)
    if text == 'Password' or text == 'Confirm Password':
        widget.config(show='*')


def widget_focus_out(event, widget, text): # this allows the user to actually enter their details and
see what they are entering
    if widget.get() == '':
        widget.insert(0, text)
    if text == 'Password' or text == 'Confirm Password':
        if widget.get() in ('Password', 'Confirm Password', ''): # shows user what they are typing
            widget.config(show='')


class GUI(Tk): #user interface class
    def __init__(self, **kwargs):
        super(GUI, self).__init__(**kwargs)
        self.font_name = 'Comic Sans' # sets a universal font that can be called
        self.font_size = 12
        self._1st_player_logged = False
        self._2nd_player_logged = False
        self.font = (self.font_name, self.font_size)
        self.font_bold = (self.font_name, self.font_size, 'bold')
        # for open tkinter window at center point
        w = 1000  # Width
```

```
        h = 600  # Height

        SCREEN_WIDTH = self.winfo_screenwidth()  # Width of the screen
        SCREEN_HEIGHT = self.winfo_screenheight()  # Height of the screen

        # Calculate Starting X and Y coordinates for Window
        x = (SCREEN_WIDTH / 2) - (w / 2)
        y = (SCREEN_HEIGHT / 2) - (h / 2)

        self.geometry('%dx%d+%d+%d' % (w, h, x, y))
        ##variables for the placment of the widgets and backgrounds##
        self.title("KMK-Fighting")
        self.resizable(False, False)
        self.Menu_Page()  # Menu_Page()
        self.Character_Selection()
        self.Login()
        self.Signup()
        self.character_images = []
        self.load_character_images()
        self.mainloop()

    def Menu_Page(self):# the main menu page
        self.menu_frame = Frame(self)
        self.menu_frame.pack(fill=BOTH, expand=True)
        self.bg_image = PhotoImage(file='assets/images/bg1.png')
        Label(self.menu_frame, image=self.bg_image).place(x=-2, y=-2)

        ##Play Button##
```

```
        Button(self.menu_frame, text='Play', font=self.font_bold, fg='White', bg='gray40', width=13,
               command=lambda: [self.menu_frame.pack_forget(), self.login_frame.pack(fill=BOTH,
expand=1)]).place(
            x=635, y=215)
         ##Signup button and placement##
        Button(self.menu_frame, text='Sign Up', font=self.font_bold, fg='White', bg='gray40',
width=13,
               command=lambda: [self.menu_frame.pack_forget(), self.signup_frame.pack(fill=BOTH,
expand=1)]).place(
            x=635, y=280)
        ##Recent Fights that have taken place##
        Button(self.menu_frame, text='Recent Fights', font=self.font_bold, fg='White', bg='gray40',
width=13,
               command=lambda: messagebox.showinfo("Recent Fights", db.get_leader_board())).place(
            x=635, y=345)

        Button(self.menu_frame, text='Exit', font=self.font_bold, fg='White', bg='gray40', width=13,
               command=lambda: sys.exit()).place(x=635, y=410)

    def Character_Selection(self):
        self.select_charater = Frame(self)
        # self.select_charater.pack(fill=BOTH, expand=1)
        self.sc_img = PhotoImage(file='assets/images/cs.png')
        Label(self.select_charater, image=self.sc_img).place(x=-2, y=-2)
        Button(self.select_charater, text='Start', width=10, bg='#1c74f7', fg='White',
               font=self.font_bold, command=lambda: self.start_game()).place(x=425, y=524)#start
button for game
        Button(self.select_charater, text='Back', width=10, bg='#1c74f7', fg='White',
               font=self.font_bold, command=lambda: [self.select_charater.pack_forget(),
```

```
                                                    self.login_frame.pack(fill=BOTH,
expand=1)]).place(x=425, y=564)# this button takes users back to the previous screen

        player_one_frame = Frame(self.select_charater, bg='#1c74f7', width=300) # select frame for
player 1
        player_one_frame.pack(side=LEFT, fill=Y)
        player_one_frame.propagate(False) # changes it to false
        player_two_frame = Frame(self.select_charater, width=300, bg='#1c74f7')
        player_two_frame.pack(side=RIGHT, fill=Y)
        player_two_frame.propagate(False) # changes it to false

        Label(player_one_frame, text='Player 1', font=(self.font_name, self.font_size + 8, 'bold'),
fg='White',
              bg='#1c74f7').pack(pady=30)
        self.player_one = ttk.Combobox(player_one_frame, width=12, font=self.font, state='readonly',
                                       values=['Human', 'Huntress', 'King', 'Samurai', 'Warrior',
'Wizard'])  # a list of the characters is displayed
        self.player_one.pack(pady=(0, 10))
        self.player_one.bind("<<ComboboxSelected>>", self.preview_character)

        self.player_one_img = Label(player_one_frame, image='', bg='#1c74f7')
        self.player_one_img.pack(pady=0)

        Label(player_two_frame, text='Player 2', font=(self.font_name, self.font_size + 8, 'bold'),
fg='White',
              bg='#1c74f7').pack(side=BOTTOM, pady=30)
        self.player_two = ttk.Combobox(player_two_frame, width=12, font=self.font, state='readonly',
                                       values=['Human', 'Huntress', 'King', 'Samurai', 'Warrior',
'Wizard'])
```

```
        self.player_two.pack(side=BOTTOM, pady=(0, 10))
        self.player_two.bind("<<ComboboxSelected>>", self.preview_character)
        self.player_two_img = Label(player_two_frame, image='', bg='#1c74f7') # this part is for
player 2 side of the
        self.player_two_img.pack(side=BOTTOM, pady=10)

    def preview_character(self, event): # function to actually display the character on the screen
        if self.player_two.get() != '':
            self.player_two_img.config(image=self.character_images[self.player_two.current()])
        if self.player_one.get() != '':
            self.player_one_img.config(image=self.character_images[self.player_one.current()])
#configures the images

    def load_character_images(self): # the character images are loaded onto the list
        path_list = os.listdir('assets/images/characters')
        print(path_list.sort())
        for cnt in path_list:
            if cnt.endswith(".png"):
                img = PhotoImage(file='assets/images/characters/' + cnt)
                self.character_images.append(img)
            print(cnt) # the list on how the the characters are presented

    def start_game(self): # the game begins after the user enters their characters
        p1 = self.player_one.get()
        p2 = self.player_two.get()
        if p1 != '':
            if p2 != '':
                p1_index = self.player_one.current()
                p2_index = self.player_two.current()
```

```
                bg_image_index = random.choice(range(3))
                self.withdraw()
                run_fighter_game(p1_index, p2_index, bg_image_index,
                                 self.username_field_1st.get(), self.username_field_2nd.get(), db)
                self.deiconify()

            else:
                messagebox.showwarning('Warning', "Player 2 has not selected any character yet ") #
error message
        else:
            messagebox.showwarning('Warning', "Player 1 has not selected any character yet")

    # Login window GUI code
    def Login(self):
        self.login_frame = Frame(self)  # creates a new frame widget for the login screen

        self.login_img = PhotoImage(file='assets/images/login.png') # loads an image to be used in the
login screen
        Label(self.login_frame, image=self.login_img).place(x=-2, y=-2)
        Button(self.login_frame, text='Start', width=10, bg='#1c74f7', fg='White',
               font=self.font_bold, command=lambda: self.check_both_logged_in()).place(x=425, y=524)
# creates a 'Start' button to start the game
        Button(self.login_frame, text='Back', width=10, bg='#1c74f7', fg='White',
               font=self.font_bold, command=lambda: [self.login_frame.pack_forget(),
                                                     self.menu_frame.pack(fill=BOTH,
expand=1)]).place(x=425, y=564) # creates a 'Back' button to return to the main menu

        player_one_frame = Frame(self.login_frame, bg='#1c74f7', width=300)
```

```
        player_one_frame.pack(side=LEFT, fill=Y) # positions the first player's frame on the left side
of the login screen
        player_one_frame.propagate(False)
        player_two_frame = Frame(self.login_frame, width=300, bg='#1c74f7') # creates a frame for the
second player's login information
        player_two_frame.pack(side=RIGHT, fill=Y)
        player_two_frame.propagate(False) # prevents the player_two_frame from automatically resizing

        self._1st_login_frame = Frame(player_one_frame, bg='#1c74f7')
        self._1st_login_frame.pack(pady=150)
        self.login_frame_1 = Frame(self._1st_login_frame, bg='#1c74f7') # creates a sub-frame for the
first player's username and password
        self.login_frame_1.pack()
        Label(self.login_frame_1, text='Player 1 Login', bg='#1c74f7', fg='White',
              font=(self.font_name, self.font_size + 7, 'bold')).pack(pady=20) # creates a label for
the first player's login information

        self.username_field_1st = Entry(self.login_frame_1, width=20, font=('Microsoft Yahei UI
Light', 11, 'bold'),
                                        bd=0,
                                        fg='#4169E1')
        self.username_field_1st.pack() # packs the username entry widget into the login_frame_1
sub-frame
        self.username_field_1st.insert(0, 'Username')
        self.username_field_1st.bind('<FocusIn>',
                                     lambda event, a=self.username_field_1st, b='Username':
widget_enter(event, a, b))
        self.username_field_1st.bind('<FocusOut>',
```

```
                                    lambda event, a=self.username_field_1st, b='Username':
widget_focus_out(event, a,

b))  # binds the 'FocusIn' event to a function

        Frame(self.login_frame_1, width=168, height=2, bg='#4169E1').pack(pady=(0, 10)) # fram for
player 1

        self.password_field_1st = Entry(self.login_frame_1, width=20, font=('Microsoft Yahei UI
Light', 11, 'bold'),
                                        bd=0,
                                        fg='#4169E1') # password entry field
        self.password_field_1st.pack() # packs the password  into the parent fram
        self.password_field_1st.insert(0, 'Password')
        self.password_field_1st.bind('<FocusIn>',
                                    lambda event, a=self.password_field_1st, b='Password':
widget_enter(event, a, b)) # binds the function to the field when it clicked

        self.password_field_1st.bind('<FocusOut>',
                                    lambda event, a=self.password_field_1st, b='Password':
widget_focus_out(event, a,

b))

        Frame(self.login_frame_1, width=168, height=2, bg='#4169E1').pack() # a line is created to
mimic a frame

        Button(self.login_frame_1, text='Login', font=('Open Sans', 16, 'bold'), fg='white',
               bg='#4169E1',
```

```
                activeforeground='white', activebackground='#4169E1', cursor='hand2', bd=0, width=12,
                command=lambda: self.sign_in(0)).pack(
            pady=10) # player 1 login button
        Button(self.login_frame_1, text='Forgot Password?', bd=0, bg='#1c74f7',
activebackground='white',
                cursor='hand2',
                font=('Microsoft Yahei UI Light', 9, 'bold'), fg='#4169E1', activeforeground='#4169E1',
                command=lambda: self.forget_pass(1)).pack() #

        # player 2 login
        self._2nd_login_frame = Frame(player_two_frame, bg='#1c74f7') # frame on the right
        self._2nd_login_frame.pack(pady=150)
        self.login_frame_2 = Frame(self._2nd_login_frame, bg='#1c74f7')
        self.login_frame_2.pack()
        Label(self.login_frame_2, text='Player 2 Login', bg='#1c74f7', fg='White',
              font=(self.font_name, self.font_size + 7, 'bold')).pack(pady=20) # player 2 login title

        self.username_field_2nd = Entry(self.login_frame_2, width=20, font=('Microsoft Yahei UI
Light', 11, 'bold'),
                                        bd=0,
                                        fg='#4169E1') # username field for player 2
        self.username_field_2nd.pack()
        self.username_field_2nd.insert(0, 'Username')
        self.username_field_2nd.bind('<FocusIn>',
                                     lambda event, a=self.username_field_2nd, b='Username':
widget_enter(event, a, b))
        self.username_field_2nd.bind('<FocusOut>',
                                     lambda event, a=self.username_field_2nd, b='Username':
widget_focus_out(event, a,
```

```
b)) # function is binded to the fields

        Frame(self.login_frame_2, width=168, height=2, bg='#4169E1').pack(pady=(0, 10))

        self.password_field_2nd = Entry(self.login_frame_2, width=20, font=('Microsoft Yahei UI
Light', 11, 'bold'),
                                        bd=0,
                                        fg='#4169E1') # password entry field for player 2
        self.password_field_2nd.pack()
        self.password_field_2nd.insert(0, 'Password')
        self.password_field_2nd.bind('<FocusIn>',
                                     lambda event, a=self.password_field_2nd, b='Password':
widget_enter(event, a, b)) # binds function to the password field

        self.password_field_2nd.bind('<FocusOut>',
                                     lambda event, a=self.password_field_2nd, b='Password':
widget_focus_out(event, a,

b))

        Frame(self.login_frame_2, width=168, height=2, bg='#4169E1').pack() # packs the player 2 login
frame into the main login page frame

        Button(self.login_frame_2, text='Login', font=('Open Sans', 16, 'bold'), fg='white',
               bg='#4169E1',
               activeforeground='white', activebackground='#4169E1', cursor='hand2', bd=0, width=12,
               command=lambda: self.sign_in(1)).pack(pady=10)
```

```
        Button(self.login_frame_2, text='Forgot Password?', bd=0, bg='#1c74f7',
activebackground='white',
                cursor='hand2',
                font=('Microsoft Yahei UI Light', 9, 'bold'), fg='#4169E1', activeforeground='#4169E1',
                command=lambda: self.forget_pass(2)).pack() # forgot password button for player 2 as
well

    def check_both_logged_in(self): # checks if both players have logged in
        if self._1st_player_logged and self._2nd_player_logged:
            self.login_frame.pack_forget()
            self.select_charater.pack(fill=BOTH, expand=1)
        else:
            messagebox.showwarning("Warning", "Both user need to login before continue...") # tells
both players to login

    def sign_in(self, which_player):
        player = [
            [self.username_field_1st.get(), self.password_field_1st.get(), self.login_frame_1,
self._1st_login_frame,
             self._1st_player_logged], # text fields
            [self.username_field_2nd.get(), self.password_field_2nd.get(), self.login_frame_2,
self._2nd_login_frame,
             self._2nd_player_logged]]
        u_name = player[which_player][0] # assign the value of username and passwords
        pass_ = player[which_player][1]
        if u_name not in ('Username', '') and pass_ not in ('Password', ''): # checks if the username
and password are not empty fields
            get_user = db.get_user(u_name.upper())
            if get_user: #if username and password are valid, program retrieves data from database
```

```
                if get_user[2] == pass_:
                    player[which_player][2].pack_forget()
                    Label(player[which_player][3], text=str(get_user[1]) + '\nLogged in Successfully',
                          font=(self.font_name, self.font_size + 8), bg='#1c74f7',
fg='white').pack(pady=(50, 20)) # text to show logged in successfully
                    Button(player[which_player][3], text='Player Stat', font=self.font_bold,
                           command=lambda: messagebox.showinfo("Player Statistics",

db.player_stat(player[which_player][0].upper()))).pack() # new window to show the stastics
                    if which_player == 0:
                        self._1st_player_logged = True
                    elif which_player == 1:
                        self._2nd_player_logged = True
                    # check both user logged in successfully
                    # print(self._1st_player_logged, self._2nd_player_logged)
                    # if self._1st_player_logged and self._2nd_player_logged:
                    #     self.login_frame.pack_forget()
                    #     self.select_charater.pack(fill=BOTH, expand=1)
                else:
                    messagebox.showwarning("Warning", "Invalid Password") # error messages
            else:
                messagebox.showwarning("Warning", "Invalid Username")
        else:
            messagebox.showwarning("Warning", "Please fill in all fields") # error to show all fields
are entered

    def forget_pass(self, which_user): #forgot password function
        if which_user == 1:
            u_name = self.username_field_1st.get() # fetches the username to reset password
```

```
        else:
            u_name = self.username_field_2nd.get() # same for player 2
        if u_name in ('Username', ''):
            messagebox.showwarning("Warning", "Username Required....") # checks if username has been
entrered so password reset can be sent
        else:
            user_res = db.get_user(u_name.upper())
            if user_res:
                response = messagebox.askyesno("Reset Password",
                                               "Click OK and new password instructions will be sent to
your Email id and will"
                                               " reset your current password.\nDo you want to
continue? ") # user prompted to press ok

                if response:

                    mail_id = user_res[-1]
                    pass_ = ''
                    for pas in range(6):
                        num = random.choice(range(0, 10))
                        pass_ += str(num)
                    msg = Send_email(mail_id, pass_)
                    if msg[-1]: # window for password reset
                        self.withdraw()
                        self.popup = Toplevel()
                        self.popup.title('KMK-Fighting (Password reset)')
                        self.popup.config(bg='#1c74f7')
                        x = (self.winfo_screenwidth() / 2) - (300 / 2)
                        y = (self.winfo_screenheight() / 2) - (300 / 2)
```

```
                        self.popup.geometry(f'300x300+{round(x)}+{round(y)}') # dimensions for the
window
                        Label(self.popup, text='Reset Password', font=(self.font_name, self.font_size
+ 12, 'bold'),
                              fg='White', bg='#1c74f7').pack(pady=20) # title
                        Label(self.popup, text='New Password', font=self.font_bold, bg='#1c74f7',
fg='White').pack()
                        self.new_pass = Entry(self.popup, font=self.font, width=20)
                        self.new_pass.pack(pady=(2, 10))
                        self.new_pass.delete(0, END)
                        Label(self.popup, text='Code', font=self.font_bold, bg='#1c74f7',
fg='White').pack()
                        self.code = Entry(self.popup, font=self.font, width=20)
                        self.code.pack(pady=(2, 10))
                        self.code.delete(0, END)
                        Button(self.popup, text='Update', font=self.font_bold,
                               command=lambda: self.reset_pass(pass_, u_name, msg[0])).pack(pady=10) #
'update' button to update password
                        self.popup.wait_window()
                        self.deiconify()
                    else:
                        messagebox.showerror("Error", msg[0])
            else:
                messagebox.showwarning("Warning", "No User found with this name...")

    def reset_pass(self, pass_, u_name, msg):
        new_password = self.new_pass.get()
        code_ = self.code.get()
        if new_password != '' and code_ != '':
```

```
            if code_ == pass_:

                res = db.update_profile(data=[str(new_password), u_name.upper()]) # opens the DB and
changes password
                print(res)
                messagebox.showinfo("Success", msg)
                self.popup.destroy()
            else:
                messagebox.showwarning('Warning', "Invalid Code") # prompts to enter the right code
        else:
            messagebox.showwarning("Warning", "Password and Code Field required")

    # Sign up GUI window code
    def Signup(self):
        self.signup_frame = Frame(self)
        # self.signup_frame.pack(fill=BOTH, expand=1)

        self.sigup_img = PhotoImage(file='assets/images/signup.png')
        Label(self.signup_frame, image=self.sigup_img).place(x=-2, y=-2)

        heading = Label(self.signup_frame, text='Sign Up', font=(self.font_name, self.font_size + 12,
'bold'),
                        bg='white',
                        fg='#4169E1') # Main title of signup page
        heading.place(x=660, y=111) # placement

        self.r_username_field = Entry(self.signup_frame, width=30, font=('Microsoft Yahei UI Light',
11, 'bold'), bd=0,
                                      fg='#4169E1') # Entry field for username 1
```

```
        self.r_username_field.place(x=590, y=170)
        self.r_username_field.insert(0, 'Username')
        self.r_username_field.bind('<FocusIn>',
                                   lambda event, a=self.r_username_field, b='Username':
widget_enter(event, a, b))
        self.r_username_field.bind('<FocusOut>',
                                   lambda event, a=self.r_username_field, b='Username':
widget_focus_out(event, a, b)) # binds the function

        Frame(self.signup_frame, width=250, height=2, bg='#4169E1').place(x=590, y=192) # signup page
frame

        self.r_password_field = Entry(self.signup_frame, width=30, font=('Microsoft Yahei UI Light',
11, 'bold'), bd=0,
                                      fg='#4169E1')
        self.r_password_field.place(x=590, y=230)
        self.r_password_field.insert(0, 'Password')
        self.r_password_field.bind('<FocusIn>',
                                   lambda event, a=self.r_password_field, b='Password':
widget_enter(event, a, b)) # same functionality for password field

        self.r_password_field.bind('<FocusOut>',
                                   lambda event, a=self.r_password_field, b='Password':
widget_focus_out(event, a, b))

        pass_frame = Frame(self.signup_frame, width=250, height=2, bg='#4169E1')  # frame for the
entry fields
        pass_frame.place(x=590, y=252)
```

```
        self.r_c_password_field = Entry(self.signup_frame, width=30, font=('Microsoft Yahei UI Light',
11, 'bold'),
                                        bd=0,
                                        fg='#4169E1')
        self.r_c_password_field.place(x=590, y=290) #placement of the password field
        self.r_c_password_field.insert(0, 'Confirm Password')
        self.r_c_password_field.bind('<FocusIn>',
                                     lambda event, a=self.r_c_password_field, b='Confirm Password':
widget_enter(event,

a, b))
        self.r_c_password_field.bind('<FocusOut>',
                                     lambda event, a=self.r_c_password_field, b='Confirm Password':
widget_focus_out(
                                         event,
                                         a, b)) # binds function to the password fields
        Frame(self.signup_frame, width=250, height=2, bg='#4169E1').place(x=590, y=312)

        self.r_email_field = Entry(self.signup_frame, width=30, font=('Microsoft Yahei UI Light', 11,
'bold'),
                                   bd=0, fg='#4169E1')
        self.r_email_field.place(x=590, y=350)
        self.r_email_field.insert(0, 'Email')
        self.r_email_field.bind('<FocusIn>', lambda event, a=self.r_email_field, b='Email':
widget_enter(event, a, b))
        self.r_email_field.bind('<FocusOut>',
                                lambda event, a=self.r_email_field, b='Email': widget_focus_out(event,
a, b))
```

```
        Frame(self.signup_frame, width=250, height=2, bg='#4169E1').place(x=590, y=372)

        Button(self.signup_frame, text='Sign Up', font=('Open Sans', 16, 'bold'), fg='white',
               bg='#4169E1',
               activeforeground='white', activebackground='#4169E1', cursor='hand2', bd=0, width=20,
               command=lambda: self.store_user_in_db()).place(x=590, y=400)

        Button(self.signup_frame, text='Back', font=('Open Sans', 16, 'bold'), fg='white',
bg='#4169E1', bd=0,
               width=20, activeforeground='white', activebackground='#4169E1', cursor='hand2',
               command=lambda: [self.signup_frame.pack_forget(), self.menu_frame.pack(fill=BOTH,
expand=1),
                                self.clear_signup_fields()]).place(
            x=590, y=440)

    def store_user_in_db(self): # function to store the new user's credentials
        username = self.r_username_field.get()
        pass_ = self.r_password_field.get() # all the variables from the window
        pass_c = self.r_c_password_field.get()
        mail = self.r_email_field.get()
        if (username != 'Username' or pass_ != 'Password' or pass_c != 'Confirm Password' or mail !=
'Email') and (
                username != '' or pass_ != '' or pass_c != '' or mail != ''): # checks if all fields
are correct
            if pass_ == pass_c:
                if email_validation(mail):
                    res = db.insert_profile([username.upper(), pass_, mail])
                    messagebox.showinfo("Success", res)
```

```
                    self.clear_signup_fields() # if passwords match and email and username have been
entered then data is stored
                else:
                    messagebox.showwarning("Warning", "Invalid Email")
            else:
                messagebox.showwarning("Warning", "Password and Confirm Password are not same...")
        else:
            messagebox.showwarning("Warning", "Fields can't be left blank") # error checking

    def clear_signup_fields(self):
        # clear fields
        self.r_username_field.delete(0, END)
        self.r_email_field.delete(0, END)
        self.r_password_field.delete(0, END)
        self.r_c_password_field.delete(0, END)
        # insert default data
        self.r_username_field.insert(0, 'Username')
        self.r_email_field.insert(0, 'Email')
        self.r_password_field.insert(0, 'Password')
        self.r_c_password_field.insert(0, 'Confirm Password')
        # don't hide text
        self.r_password_field.config(show='')
        self.r_c_password_field.config(show='')


gui = GUI()
```

```
                                        ###Database.py SCRIPT 2###
import sqlite3 as sql


# class
class DB:
    # constructor
    def __init__(self, **kwargs):
        super(DB, self).__init__(**kwargs)
        # creating or connecting to DB
        self.conn = sql.connect('kmk-fighting-db.db')
        # creating cursor for DML operation queries
        self.cur = self.conn.cursor()
        print('database connected')
        # calling function
        self.create_database_structure()

    def create_database_structure(self):
        # create table if not exist
        self.cur.execute("""Create table If not exists Users (U_id Integer primary key autoincrement,
                            Username Text NOT NULL UNIQUE,
                            Password Text Not Null,
                            Email Text Not Null UNIQUE)""")

        # create table Leaderboard
```

```
        self.cur.execute("""Create table If not exists Leaderboard (b_id Integer primary key
autoincrement,
                    Round_no Text Not Null,
                    play_time Text Not Null,
                    Player1_Name Text Not Null,
                    Player2_Name Text Not Null,
                    win Text Not Null)""")

    # fetch user by giving the username
    def get_user(self, username):
        # select query
        self.cur.execute("Select * from Users where username=?", (username,))
        # get result
        res = self.cur.fetchall()

        if res:
            # return result if there is any
            return res[0]
        else:
            # else return none
            return None

    # get Leader board info function
    def get_leader_board(self):
        msg = ''
        # select query
        self.cur.execute('Select * from Leaderboard')
        # get data from DB
        res = self.cur.fetchall()
```

```
        if res:
            for i in res:
                msg += 'Round ' + str(i[1]) + ' --> ' + i[3] + ' vs ' + i[4] + ' Winner(' + str(i[5])
+ ')\n'
        else:
            msg = 'No data Found'
        # return result
        return msg

    # player stat
    def player_stat(self, p_name):
        sum_time = 0
        avg_time = 0
        tol_rounds = 0
        msg = 'No Data Found'
        self.cur.execute(
            "select sum(play_time/60), avg(play_time/60), count(round_no) from Leaderboard WHERE
(player1_name = ? or player2_name= ?)", # aggregate SQL functions
            (p_name, p_name))
        res = self.cur.fetchall()
        self.cur.execute("select COUNT(win) from Leaderboard WHERE win=?", (p_name,))
        res1 = self.cur.fetchall()
        print(res)
        if res[0][0]:
            sum_time = round(res[0][0], 2)
            avg_time = round(res[0][1], 2)
            tol_rounds = res[0][2]
        msg = "Total Play time in seconds " + str(sum_time) + '\n\nAvg. Play time per round ' + str(
```

```
            avg_time) + '\n\nTotal Round Wins ' + str(res1[0][0]) + '/' + str(tol_rounds) # player
stats

        return msg

    # Leaderboard insert
    def insert_leaderboard(self, details):
        try:
            # insert query
            self.cur.execute("Insert into LeaderBoard Values (Null, ?, ?, ?, ?, ?)",
                             (details[0], details[1], details[2], details[3], details[4]))
            msg = 'Data Inserted successfully'
            # save DB
            self.conn.commit()
            return msg
        except Exception as e:
            return str(e)

    # update profile function
    def update_profile(self, data):
        try:
            # update query
            self.cur.execute("update Users set Password=? where username=?",
                             (data[0], data[1]))
            # save DB
            self.conn.commit()
            msg = 'Profile Updated successfully'
        except Exception as e:
            msg = str(e)
```

```
        return msg

    def insert_profile(self, data):
        try:
            # update query
            self.cur.execute("Insert into Users Values ( Null, ?, ?, ?)", (data[0], data[1], data[2]))
            # save DB
            self.conn.commit()
            msg = 'User Register successfully'
        except Exception as e:
            msg = str(e)
        return msg
```

```
                                    ###email_sender.py SCRIPT 3###
import smtplib
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText


def Send_email(mail_to, password):
    try:
        # setting email id, pass, subject, from , to and setting body msg
        email_id = 'kizzapass@gmail.com'
        email_pass = 'yjysomgntqtexhah'
        email_msg = MIMEMultipart()
        email_msg['Subject'] = 'KMK-Fighting account Reset Password.'
        email_msg['From'] = email_id
        email_msg['To'] = mail_to
        email_msg.attach(MIMEText("Your reset password request has been accepted Successfully."
                                  "\n\nYour  password reset code is '" + password +
                                  "'\n\nThis email is auto generated so kindly don't reply on it",
'plain'))

        with smtplib.SMTP('smtp.gmail.com', port=587) as smtp:
            # start TLS for security
            smtp.starttls()
```

```
            smtp.ehlo()
            # login in email
            smtp.login(email_id, email_pass)
            msg = email_msg.as_string()
            # send email
            smtp.sendmail(email_id, mail_to, msg)
            # quit server
            smtp.quit()

        return ["Password Reset Successfully...", True]
    except Exception as e:
        print(str(e))
        return [str(e), False]
```

```
                                        ###fighter.py SCRIPT 4###
import pygame


class Fighter(): # class for all the fighters
    def __init__(self, player, x, y, flip, data, sprite_sheet, animation_steps, sound, damage): #
attribute parameters
        self.player = player
        self.size = data[0] # variable/attribute for player 1
        self.image_scale = data[1]
        self.offset = data[2]
        self.flip = flip # flips to face the opponent
        self.animation_list = self.char_imgs(sprite_sheet, animation_steps)
        self.action = 0  #list actions 0:idle #1:run #2:jump #3:attack1 #4: attack2 #5:hit #6:death
        self.frame_index = 0
        self.image = self.animation_list[self.action][self.frame_index]
        self.update_time = pygame.time.get_ticks()
        self.rect = pygame.Rect((x, y, 80, 180)) # this is the character rectangle which helps us with
collisions
        self.y_velocity = 0
        self.walking = False
        self.jump = False
        self.attacking = False
        self.attack_type = 0
```

```
        self.attack_cooldown = 0
        self.attack_sound = sound # attack sound attribute
        self.attack_damage=damage
        self.hit = False
        self.player_hp = 100 # all players have a set player_hp of 100 points
        self.alive = True

    def char_imgs(self, sprite_sheet, animation_steps): # extract frames and images from the
corresponding character

        animation_list = []
        for y, animation in enumerate(animation_steps):
            temp_img_list = []
            for x in range(animation):
                temp_img = sprite_sheet.subsurface(x * self.size, y * self.size, self.size, self.size)
# this is a temporary image that is cycled through sprite sheet and then moves to next frame
                temp_img_list.append(
                    pygame.transform.scale(temp_img, (self.size * self.image_scale, self.size *
self.image_scale))) # use of list operations
            animation_list.append(temp_img_list)
        return animation_list

    def move(self, screen_width, screen_height, surface, target, round_over): # movement method
        PLAYER_SPEED = 10 # set speed
        GRAVITY = 2 # set gravity
        dx = 0 # the shift in the x direction and y direction
        dy = 0
        self.walking = False
        self.attack_type = 0
```

```
        key = pygame.key.get_pressed() # allows users to press keys


        if self.attacking == False and self.alive == True and round_over == False: # ensures that
users can only perform other actions if not opponent currently attacking
            # check player 1 controls
            if self.player == 1:

                #forward and back movement#
                if key[pygame.K_a]: # pressing the 'a' key will make the player speed variable
negative hence character moves to the left
                    dx = -PLAYER_SPEED # change dx variable
                    self.walking = True
                if key[pygame.K_d]:
                    dx = PLAYER_SPEED # pressing the 'd' key will make the player speed variable
positive hence character moves to the right
                    self.walking = True

                #jumping#
                if key[pygame.K_w] and self.jump == False:
                    self.y_velocity = -30 # y velolity determines how quick the character can jump
                    self.jump = True # player can jump

                ##attack animations##
                if key[pygame.K_r] or key[pygame.K_t]:
                    self.attack(target)
```

```
            if key[pygame.K_r]:
                self.attack_type = 1 # determine whether user wants to use attack 1 or 2
            if key[pygame.K_t]:
                self.attack_type = 2

        #player 2 controls#
        if self.player == 2:
            # forward backward movement
            if key[pygame.K_LEFT]: # same as above but now arrow keys are being used
                dx = -PLAYER_SPEED
                self.walking = True # same principles
            if key[pygame.K_RIGHT]:
                dx = PLAYER_SPEED
                self.walking = True
            #jumping#
            if key[pygame.K_UP] and self.jump == False:
                self.y_velocity = -30
                self.jump = True
            #attack#
            if key[pygame.K_KP1] or key[pygame.K_KP2]: # numpad 1 and 2
                self.attack(target)

                if key[pygame.K_KP1]:
                    self.attack_type = 1
                if key[pygame.K_KP2]:
                    self.attack_type = 2
```

```
        self.y_velocity += GRAVITY # gravity is nothing but a if statement which changes the y
position of character if it goes above a certain thereshold
        dy += self.y_velocity

        # ensure both stays on game window
        if self.rect.left + dx < 0:
            dx = -self.rect.left
        if self.rect.right + dx > screen_width:
            dx = screen_width - self.rect.right # game is run on invisible rectangles. this concept is
used for collisions later too.
        if self.rect.bottom + dy > screen_height - 110:
            self.y_velocity = 0
            self.jump = False
            dy = screen_height - 110 - self.rect.bottom #this case the invisible rectangle is checked
if it is touching the game window boundary


        if target.rect.centerx > self.rect.centerx: # this time the centre of the invisible is
compared to face off the two players at all times
            self.flip = False
        else:
            self.flip = True # variable changes to true


        if self.attack_cooldown > 0:
            self.attack_cooldown -= 1 # attack cooldown is initiated to stop players from spamming


        self.rect.x += dx
```

```
        self.rect.y += dy # the rectangles move with the players


    def update(self): # update method
        # check what action the player is performing
        if self.player_hp <= 0:
            self.player_hp = 0
            self.alive = False # the player has been defeated
            self.update_action(6)  # 6:Death animation is initiated
        elif self.hit:
            self.update_action(5)  # 5:hit animation is initiated
        elif self.attacking:
            if self.attack_type == 1:
                self.update_action(3)  # 3:attack1
            elif self.attack_type == 2:
                self.update_action(4)  # 4:attack2
        elif self.jump:
            self.update_action(2)  # 2:jump
        elif self.walking:
            self.update_action(1)  # 1:run
        else:
            self.update_action(0)  # 0:idle

        animation_cooldown = 50
        # update image
        self.image = self.animation_list[self.action][self.frame_index]
        # error check for cooldowns
        if pygame.time.get_ticks() - self.update_time > animation_cooldown:
            self.frame_index += 1
```

```
            self.update_time = pygame.time.get_ticks()

        if self.frame_index >= len(self.animation_list[self.action]):

            if not self.alive:
                self.frame_index = len(self.animation_list[self.action]) - 1  # End animations
            else:
                self.frame_index = 0
                # check if an attack was executed
                if self.action == 3 or self.action == 4:
                    self.attacking = False
                    self.attack_cooldown = 20
                # check if damage was taken
                if self.action == 5:
                    self.hit = False
                    # if the player was in the middle of an attack, then the attack is stopped
                    self.attacking = False
                    self.attack_cooldown = 20

    def attack(self, target):
        if self.attack_cooldown == 0:
            self.attacking = True
            self.attack_sound.play()
            attack_rectangle = pygame.Rect(self.rect.centerx - (2 * self.rect.width * self.flip),
self.rect.y,
                                           2 * self.rect.width, self.rect.height) # attack rectangle to
check for collsions
            if attack_rectangle.colliderect(target.rect):
                target.player_hp -= self.attack_damage
```

```
                target.hit = True

    def update_action(self, new_action):
        # check if the new action is different to the previous one
        if new_action != self.action:
            self.action = new_action
            # update the animation settings
            self.frame_index = 0
            self.update_time = pygame.time.get_ticks()

    def draw(self, surface):
        img = pygame.transform.flip(self.image, self.flip, False) #object's image and a boolean value
for whether or not to flip the image. This creates a new transformed image, which is stored in the img
variable.
        surface.blit(img, (
        self.rect.x - (self.offset[0] * self.image_scale), self.rect.y - (self.offset[1] *
self.image_scale)))
```

```
                                        ###main.py SCRIPT 5###
import pygame
from pygame import mixer # sound module
from fighter import Fighter # fighter class is imported
from tkinter import messagebox

winner = ''


def run_fighter_game(player1, player2, stage, p1_name, p2_name, db): # this function is called in the
main menu file after each user has logged in
    mixer.init()
    pygame.init()

    GW_Width = 1000 # game window width
    GW_Height = 600 # game window height

    gw = pygame.display.set_mode((GW_Width, GW_Height)) # game window variable
    pygame.display.set_caption("KMK-Fighting")

    ##Frame rate of the game##
    clock = pygame.time.Clock()
    FPS = 60
```

```
# define colours
RED = (255, 0, 0)
GREEN = (0, 255, 0)
WHITE = (255, 255, 255)

# countdown and cooldown variables
intro_count = 3
last_count_update = pygame.time.get_ticks()
player_score = [0, 0]  # player scores. [P1, P2]
round_num = 0
round_over = False
CoolDown = 2000

##Character Variables##
##CHARACTER_INITIAL (size) = 162##
##CHARACTER_SCALE = 4##
##CHARACTER_OFFSET = [72, 56]## positioning ##
##CHARACTER_DATA = [character_INITIAL, character_SCALE, character_OFFSET]##
##Character Variables##

##Warrior Variables##
WARRIOR_INITIAL = 162
WARRIOR_SCALE = 4
WARRIOR_OFFSET = [72, 56]
WARRIOR_DATA = [WARRIOR_INITIAL, WARRIOR_SCALE, WARRIOR_OFFSET]
##Warrior Variables##

##Wizard Variables##
```

```
WIZARD_INITIAL = 190
WIZARD_SCALE = 2
WIZARD_OFFSET = [72, 44]
WIZARD_DATA = [WIZARD_INITIAL, WIZARD_SCALE, WIZARD_OFFSET]
##Wizard Variables##

##Human Variables##
HUMAN_INITIAL = 126
HUMAN_SCALE = 3.5
HUMAN_OFFSET = [62, 34]
HUMAN_DATA = [HUMAN_INITIAL, HUMAN_SCALE, HUMAN_OFFSET]
##Human Variables##

##Samurai Variables##
SAMURAI_INITIAL = 200
SAMURAI_SCALE = 3
SAMURAI_OFFSET = [72, 63]
SAMURAI_DATA = [SAMURAI_INITIAL, SAMURAI_SCALE, SAMURAI_OFFSET]
##Samurai Variables##

##Huntress Variables##
HUNTRESS_INITIAL = 150
HUNTRESS_SCALE = 4
HUNTRESS_OFFSET = [55, 49]
HUNTRESS_DATA = [HUNTRESS_INITIAL, HUNTRESS_SCALE, HUNTRESS_OFFSET]
##Huntress Variables##

##King Variables##
KING_INITIAL = 155
```

```
    KING_SCALE = 2
    KING_OFFSET = [72, 30]
    KING_DATA = [KING_INITIAL, KING_SCALE, KING_OFFSET]
    ##King Variables##



    pygame.mixer.music.load("assets/audio/BossBattle_0085_BPM111_Bbm_Duel_L.mp3")
    pygame.mixer.music.set_volume(0.5)
    pygame.mixer.music.play(-1, 0.0) #duration and loops music infinitly
    sword_fx = pygame.mixer.Sound("assets/audio/sword.wav")
    sword_fx.set_volume(0.5)
    magic_fx = pygame.mixer.Sound("assets/audio/magic.wav")
    magic_fx.set_volume(0.75)



    background_img = pygame.image.load("assets/images/background/background" + str(stage) +
".png").convert_alpha() #loads background images depending on stage

    ## load spritesheets for each of my 6 characters ##
    human_sheet = pygame.image.load("assets/images/human/Sprites/Human1.png").convert_alpha()
    samurai_sheet = pygame.image.load("assets/images/Samurai/Sprites/samurai.png").convert_alpha()
    huntress_sheet = pygame.image.load("assets/images/Huntress/Sprites/huntress.png").convert_alpha()
# pygame.load module
    king_sheet = pygame.image.load("assets/images/King/Sprites/king.png").convert_alpha()
    warrior_sheet = pygame.image.load("assets/images/warrior/Sprites/warrior.png").convert_alpha()
    wizard_sheet = pygame.image.load("assets/images/wizard/Sprites/wizard.png").convert_alpha()
    ## load spritesheets for each of my 6 characters ##
```

```
    win_img = pygame.image.load("assets/images/icons/victory1.png").convert_alpha() # this image is
shown after each round ends

    # use of lists to flick through sprite sheets and display the frames #
    HUMAN_ANIMATION_STEPS = [10, 8, 3, 7, 6, 3, 10]
    SAMURAI_ANIMATION_STEPS = [8, 8, 2, 6, 6, 4, 6]
    HUNTRESS_ANIMATION_STEPS = [8, 8, 2, 5, 5, 3, 8]
    KING_ANIMATION_STEPS = [6, 8, 2, 6, 6, 4, 10]
    WARRIOR_ANIMATION_STEPS = [4, 8, 2, 4, 4, 3, 8]
    WIZARD_ANIMATION_STEPS = [6, 8, 2, 8, 8, 4, 7]


    count_font = pygame.font.Font("assets/fonts/turok.ttf", 80)
    score_font = pygame.font.Font("assets/fonts/turok.ttf", 30)

    # function for drawing text
    def draw_text(text, font, text_col, x, y):
        img = font.render(text, True, text_col)
        gw.blit(img, (x, y))


    def background_blit():
        scaled_background = pygame.transform.scale(background_img, (GW_Width, GW_Height)) # function
for scaling background and displaying it
        gw.blit(scaled_background, (0, 0))

    # function for drawing fighter player_hp bars
    def construct_healthbar(player_hp, x, y):
        ratio = player_hp / 100
```

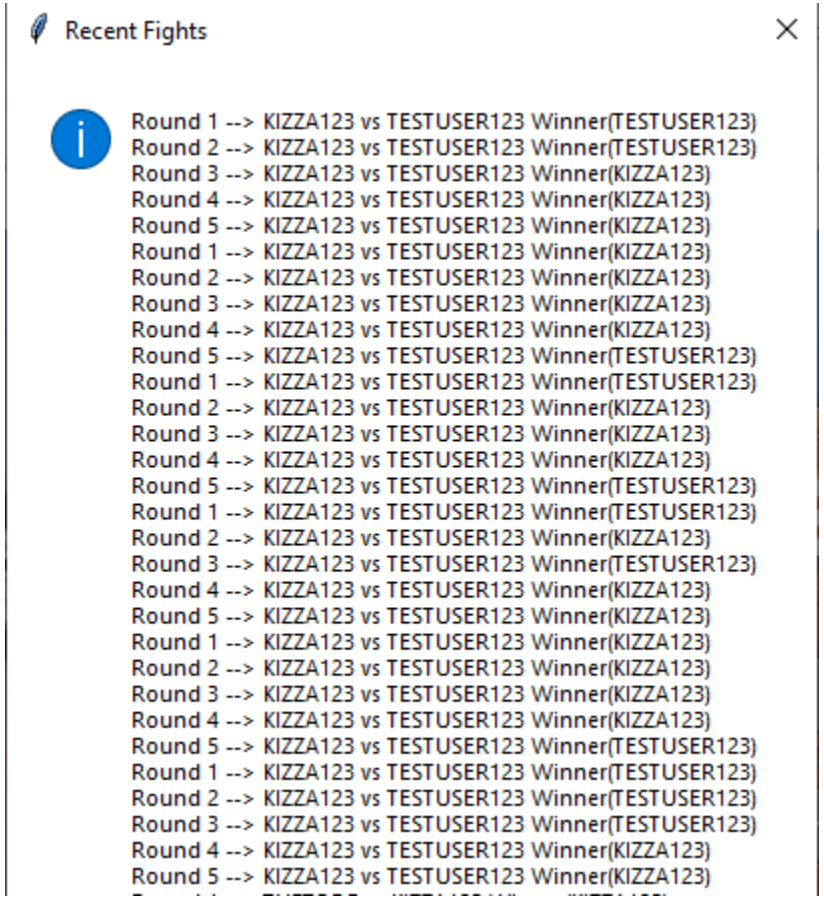
```
        pygame.draw.rect(gw, WHITE, (x - 2, y - 2, 404, 34))
        pygame.draw.rect(gw, RED, (x, y, 400, 30))
        pygame.draw.rect(gw, GREEN, (x, y, 400 * ratio, 30))

    ## instances of characters ##
    load_characters = [[HUMAN_DATA, human_sheet, HUMAN_ANIMATION_STEPS, sword_fx, 10],
                   [HUNTRESS_DATA, huntress_sheet, HUNTRESS_ANIMATION_STEPS, sword_fx, 12], #
parameters and sounds for characters
                   [KING_DATA, king_sheet, KING_ANIMATION_STEPS, sword_fx, 13],
                   [SAMURAI_DATA, samurai_sheet, SAMURAI_ANIMATION_STEPS, sword_fx, 15],
                   [WARRIOR_DATA, warrior_sheet, WARRIOR_ANIMATION_STEPS, sword_fx, 11],
                   [WIZARD_DATA, wizard_sheet, WIZARD_ANIMATION_STEPS, magic_fx, 16]]
    playerfighter_1 = Fighter(1, 200, 310, False, load_characters[player1][0],
load_characters[player1][1], load_characters[player1][2],
                        load_characters[player1][3], load_characters[player1][4]) # Fighter class
which is seperate but loads in all the attributes for a chosen character
    playerfighter_2 = Fighter(2, 700, 310, True, load_characters[player2][0],
load_characters[player2][1], load_characters[player2][2],
                        load_characters[player2][3], load_characters[player2][4])

    ###MAIN GAME###
    run = True
    while run:

        clock.tick(FPS) # frame rates
        if round_num == 5:
            run = False
            if player_score[0] > player_score[1]: # player_score variables
                winner = 'Player 1 Wins'
```

```
            else:
                winner = 'Player 2 Wins'
            messagebox.showinfo("Winner", winner)

        background_blit()


        construct_healthbar(playerfighter_1.player_hp, 20, 20)
        construct_healthbar(playerfighter_2.player_hp, 580, 20) # position of player_hp bars
        draw_text("P1: " + str(player_score[0]), score_font, RED, 20, 60)
        draw_text("P2: " + str(player_score[1]), score_font, RED, 580, 60) # position of player
player_score

        # countdown #
        if intro_count <= 0:
            # fighter parameters #
            playerfighter_1.move(GW_Width, GW_Height, gw, playerfighter_2, round_over)
            playerfighter_2.move(GW_Width, GW_Height, gw, playerfighter_1, round_over)
        else:
            # display count timer
            draw_text(str(intro_count), count_font, RED, GW_Width / 2, GW_Height / 3)
            # update count timer
            if (pygame.time.get_ticks() - last_count_update) >= 1000:
                intro_count -= 1
                last_count_update = pygame.time.get_ticks()


        playerfighter_1.update()
        playerfighter_2.update()
```

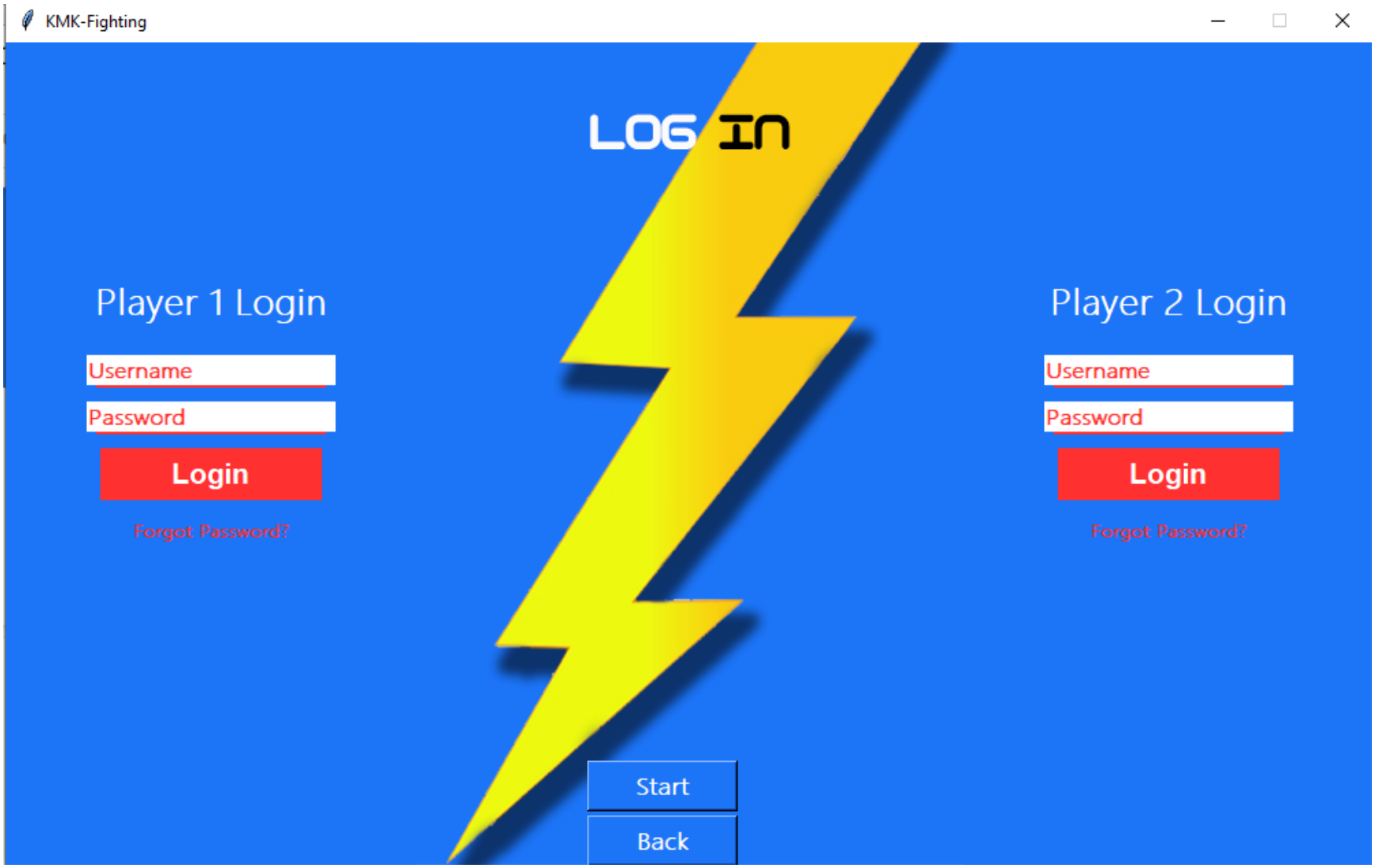
```
        playerfighter_1.draw(gw)
        playerfighter_2.draw(gw) # loads the fighter of the corresponding player on the background

        # check for player defeat
        if not round_over: # statement to check if round has finished
            if not playerfighter_1.alive:
                player_score[1] += 1 # increments player_score
                round_over = True
                round_over_time = pygame.time.get_ticks()
                round_num += 1 #next round begins
                time = round((round_over_time / 1000), 1)
                db.insert_leaderboard([round_num, time, p1_name.upper(), p2_name.upper(),
p2_name.upper()])# inserts the winner of round in recent fights


            elif not playerfighter_2.alive:
                player_score[0] += 1 # increments player_score
                round_over = True
                round_over_time = pygame.time.get_ticks()
                round_num += 1 #next round begins
                time = round((round_over_time / 1000), 1)
                db.insert_leaderboard([round_num, time, p1_name.upper(), p2_name.upper(),
p1_name.upper()]) # inserts the winner of round in recent fights


        else:
```

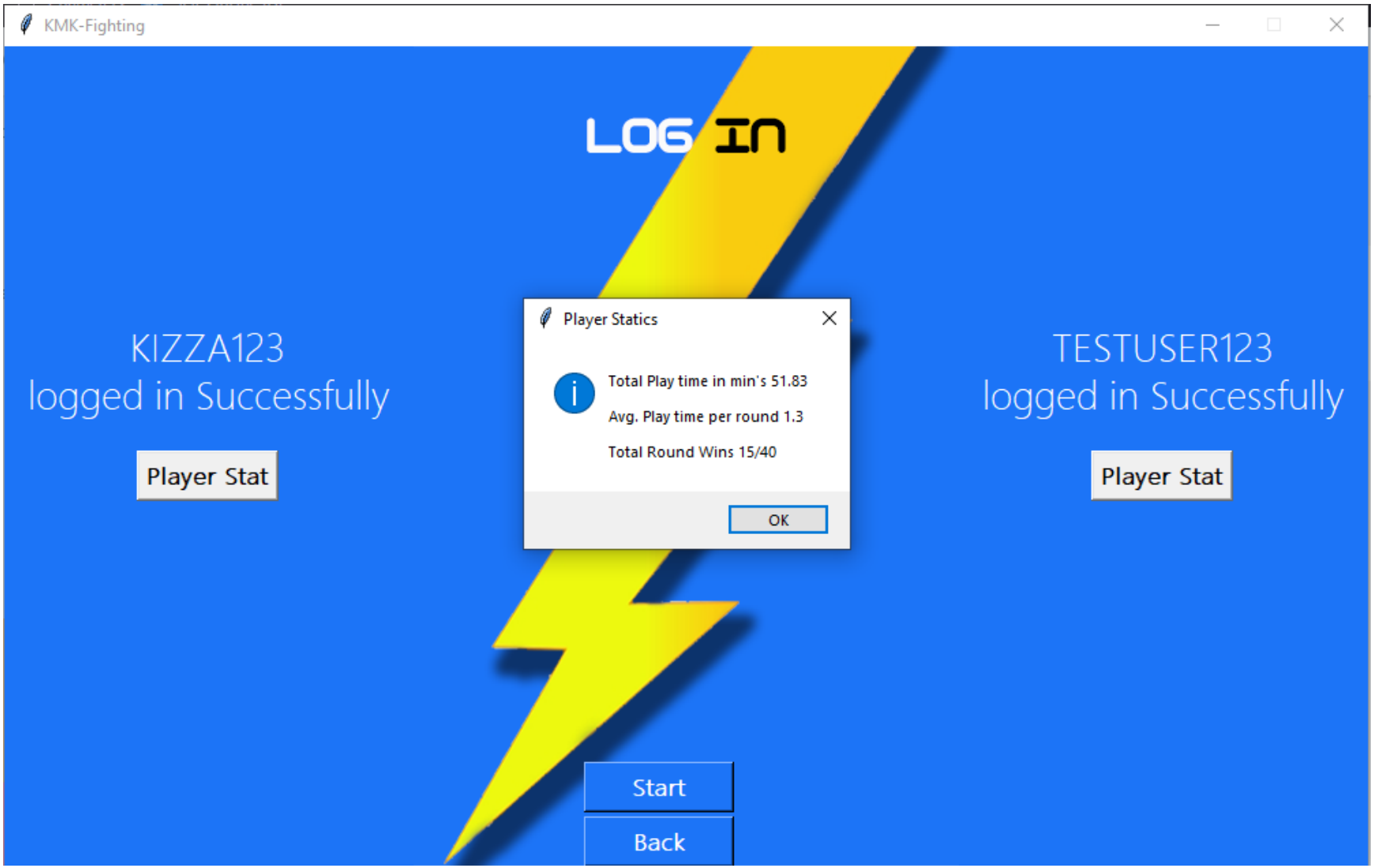
```
            gw.blit(win_img, (360, 150)) # position of victory image
            if pygame.time.get_ticks() - round_over_time > CoolDown:
                round_over = False # checks if 5 rounds have been played
                intro_count = 3
                playerfighter_1 = Fighter(1, 200, 310, False, load_characters[player1][0],
load_characters[player1][1],
                                          load_characters[player1][2],
                                          load_characters[player1][3],
                                          load_characters[player1][4])
                playerfighter_2 = Fighter(2, 700, 310, True, load_characters[player2][0],
load_characters[player2][1],
                                          load_characters[player2][2],
                                          load_characters[player2][3],
                                          load_characters[player2][4])

        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                run = False


        pygame.display.update() # updates display


    pygame.quit()
```

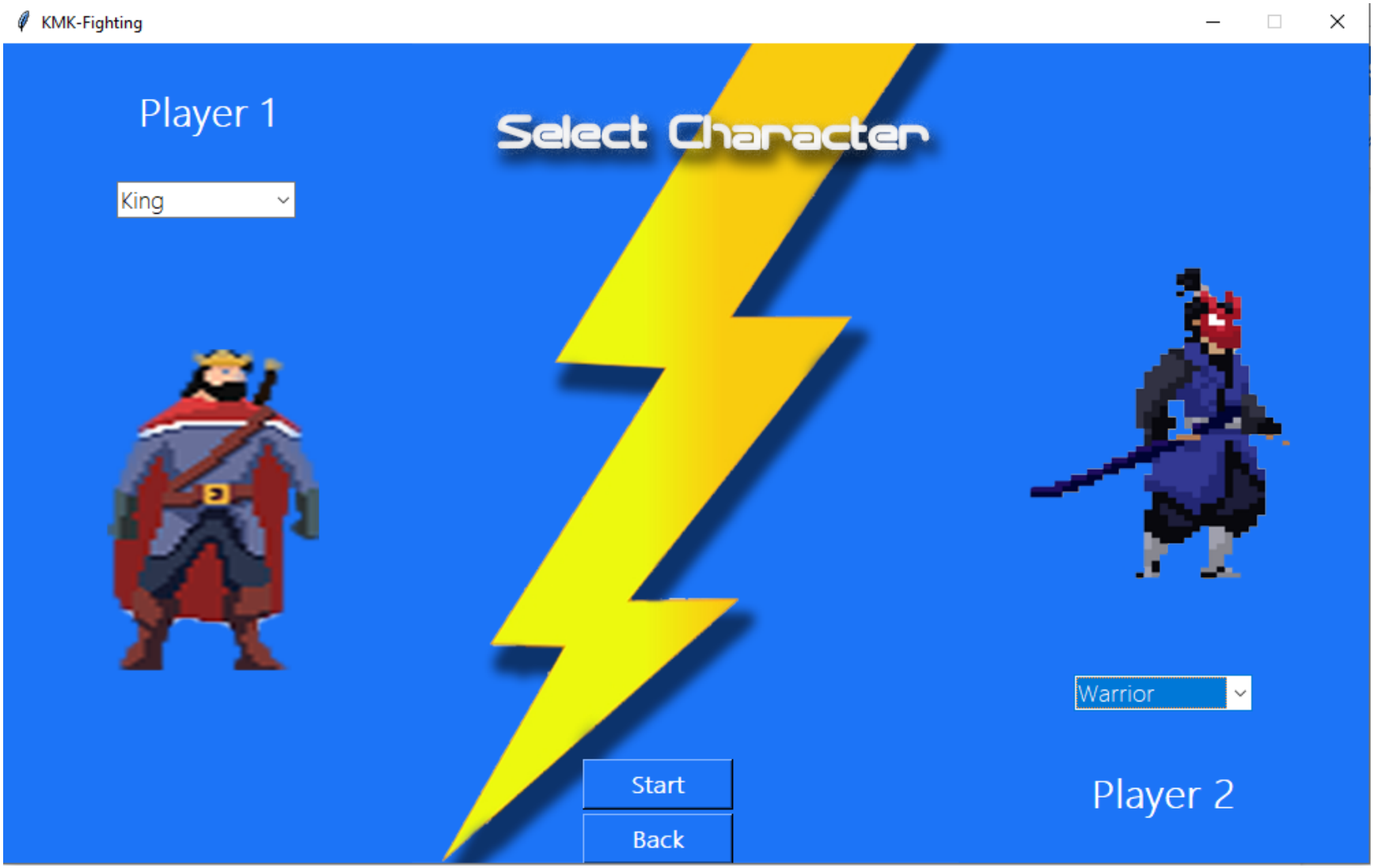
FINAL PRODUCT


Recent Fights
Round 1 --> KIZZA123 vs TESTUSER123 Winner(TESTUSER123)
Round 2 --> KIZZA123 vs TESTUSER123 Winner(TESTUSER123)
Round 3 --> KIZZA123 vs TESTUSER123 Winner(KIZZA123)
Round 4 --> KIZZA123 vs TESTUSER123 Winner(KIZZA123)
Round 5 --> KIZZA123 vs TESTUSER123 Winner(KIZZA123)
Round 1 --> KIZZA123 vs TESTUSER123 Winner(KIZZA123)
Round 2 --> KIZZA123 vs TESTUSER123 Winner(KIZZA123)
Round 3 --> KIZZA123 vs TESTUSER123 Winner(KIZZA123)
Round 4 --> KIZZA123 vs TESTUSER123 Winner(KIZZA123)
Round 5 --> KIZZA123 vs TESTUSER123 Winner(TESTUSER123)
Round 1 --> KIZZA123 vs TESTUSER123 Winner(TESTUSER123)
Round 2 --> KIZZA123 vs TESTUSER123 Winner(KIZZA123)
Round 3 --> KIZZA123 vs TESTUSER123 Winner(KIZZA123)
Round 4 --> KIZZA123 vs TESTUSER123 Winner(KIZZA123)
Round 5 --> KIZZA123 vs TESTUSER123 Winner(TESTUSER123)
Round 1 --> KIZZA123 vs TESTUSER123 Winner(TESTUSER123)
Round 2 --> KIZZA123 vs TESTUSER123 Winner(KIZZA123)
Round 3 --> KIZZA123 vs TESTUSER123 Winner(TESTUSER123)
Round 4 --> KIZZA123 vs TESTUSER123 Winner(KIZZA123)
Round 5 --> KIZZA123 vs TESTUSER123 Winner(KIZZA123)
Round 1 --> KIZZA123 vs TESTUSER123 Winner(KIZZA123)
Round 2 --> KIZZA123 vs TESTUSER123 Winner(KIZZA123)
Round 3 --> KIZZA123 vs TESTUSER123 Winner(KIZZA123)
Round 4 --> KIZZA123 vs TESTUSER123 Winner(KIZZA123)
Round 5 --> KIZZA123 vs TESTUSER123 Winner(TESTUSER123)
Round 1 --> KIZZA123 vs TESTUSER123 Winner(TESTUSER123)
Round 2 --> KIZZA123 vs TESTUSER123 Winner(TESTUSER123)
Round 3 --> KIZZA123 vs TESTUSER123 Winner(TESTUSER123)
Round 4 --> KIZZA123 vs TESTUSER123 Winner(KIZZA123)
Round 5 --> KIZZA123 vs TESTUSER123 Winner(KIZZA123)


KMK-Fighting
KMK
FIGHTING
Sign Up
Username
Password
Confirm Password
Email
Sign Up
Back


KMK-Fighting
LOG IN
Player 1 Login
Username
Password
Login
Forgot Password?
Player 2 Login
Username
Password
Login
Forgot Password?
Start
Back


KMK-Fighting
LOG IN
KIZZA123
logged in Successfully
Player Stat
TESTUSER123
logged in Successfully
Player Stat
Player Statics
Total Play time in min's 51.83
Avg. Play time per round 1.3
Total Round Wins 15/40
OK
Start
Back


KMK-Fighting
Player 1
Select Character
King
Warrior
Start
Back
Player 2


KMK-Fighting
P1: 0
P2: 0

Testing

Game

Main Menu Display

Test num	Test Subject	Expected Outcome	Pass/Fail	Data Type	Proof/ Video Link
A1	Initial Game menu	A background image with 4 buttons for ‘Play’, ‘Sign Up’, ‘Recent Fights’ and ‘Exit’. All are placed towards the right of the screen	PASS	Normal	1:12
A2	Play Button	Top of the menu the button to play the game is shown. The frame should be grey and the text should be white	PASS	Normal	(See proof)

A3	Sign Up	This Button is shown below the Play button to the right of the screen. The frame should be grey and the text should be white	PASS	Normal	(see proof)
A4	Recent Fights Button	This Button is shown below the Sign-Up button to the right of the screen. The frame should be grey and the text should be white	PASS	Normal	(see proof)
A5	Exit Button	Exit button with a frame is displayed	PASS	Normal	9:50

Recent Fights Functionality and principles of operation

Test ID	Test Subject	Expected Outcome	Pass/Fail	Data Type	Proof/Video Link
C1	Button interaction	Button for recent fights should be interactable for the user so that they can click it	PASS	Normal	1:19
C2	Recent fights window	This button provides the user a window which shows all the recent fights with each winner of a round	PASS	Normal	(see proof)
C3	Data fetch	The data should be fetched from an external table and be displayed	PASS	Normal	(see proof)

Sign Up page functionality and principles of operation

Test ID	Test Subject	Expected Outcome	Pass/Fail	Data Type	Video Link / Proof
D1	Button press	Button for sign up should be interactable for the user so that they can click it	PASS	Normal	3:34
D2	User window / UI	A window is displayed with a background and boxes (fields) to enter credentials for the user	PASS	Normal	(see proof)
D3	Buttons in the window	Sign Up and the Back button is displayed	PASS	Normal	(see proof)

D4	Disappearing fields/ guide	The guide for the user fields such as 'Password' Disappear when the field is clicked. These functions are intended to guide the user	PASS	Normal	3:37
D5	Fields, Buttons and Frame colour	The fields, buttons and the frame should be pink	PASS	Normal	(see proof)
D6	Password matching	Error message is presented when passwords do not match	PASS	Erroneous	(see proof)

D7	Unique Email	Error message is presented and the player cannot register if the email is already in the system	PASS	Erroneous	(See proof)
D8	Sign Up button / User credentials storage	If everything is correct, the user presses sign up button and Player credentials are stored in an external SQL table	PASS	Normal	7:09
D9	Invalid Email	Error message is presented and the player cannot register if the email is not a set syntax	PASS	Erroneous	(see proof)

Start Button functionality and principles of operation

Test ID	Test Subject	Expected Outcome	Pass/Fail	Data Type	Video Link/Proof
E1	Button interaction	Button for start should be interactable for the user so that they can click it	PASS	Normal	7:25
E2	Login Page	Pressing Play will guide users to the login page of the game	PASS	Normal	7:26

Login Page

Test ID	Test Subject	Expected Outcome	Pass/Fail	Data Type	Video Link/Proof
F1	Background	Background image is imported and shown	PASS	Normal	(See proof)

F2	User window / UI	Boxes (fields) to enter credentials for each player Username and Password. Player 1 on the left and Player 2 on the right	PASS	Normal	(See proof)
F3	Disappearing fields/ guide	The guide for the user fields such as 'Password' Disappear when the field is clicked. These functions are intended to guide the user	PASS	Normal	8:29
F4	Start button	buttons should be displayed at the bottom of the screen, and interactable to the user as well	PASS	Normal	(See proof)

F5	Reset Password	This button should be displayed at the bottom of the entry fields and interactable	PASS	Normal	(See proof)
F6	Login Page title	The title should be displayed at the very top of the screen	PASS	Normal	(See proof)
F7	Successful login (Player 1)	Message is shown that Player 1 has successfully logged in.	PASS	Normal	8:49
F8	Successful login (Player 2)	Display Player 2 has successfully logged in.	PASS	Normal	9:07

F9	Player Stat button	Player Stat button is displayed for each corresponding user	PASS	Normal	(see proof)
F10	Play button functionality	When the user presses this button, it should progress to the character selection screen	PASS	Normal	9:45

Player Stat functionality

Test ID	Test Subject	Expected Outcome	Pass/Fail	Data Type	Video Link/Proof
G1	When to display	Displayed after the corresponding user has successfully logged in	PASS	Normal	9:05
G2	Interactable	Users can press it	PASS	Normal	9:09
G3	Calculations	Program applies calculations for the player stats for the corresponding player. Calculations are stored in interlinked tables	PASS	Normal	(see proof)

G4	Functionality	After button is pressed, a window should pop up with the stats for the corresponding user	PASS	Normal	(see proof)

Password Reset functionality

Test ID	**Test Subject**	**Expected Outcome**	**Pass/Fail**	**Data Type**	**Video Link/Proof**
H1	Forgot Password?	This button is presented at the bottom of the user entry fields on the login page	PASS	Normal	(see proof)

H2	Send code	After pressing the button, an email with a reset code is sent to the corresponding email for that user	PASS	Normal	(see proof)
H3	Code received.	Users can retrieve this code from their email inbox	PASS	Normal	13:51
H4	Reset Window	A window is displayed. Code and New password fields are displayed	PASS	Normal	(see proof)
H5	Update DB	New password for the corresponding user is updated in the external database	PASS	Normal	16:03

Character Select Screen

Test ID	Test Subject	Expected Outcome	Pass/Fail	Data Type	Video Link/Proof
I1	Background Image	A background image is displayed	PASS	Normal	(see proof)
I2	2 lists on each side of the screen	These are placed on either side of the screen. They will show all the characters that can be chosen. A total of 6 different characters	PASS	Normal	9:48
I3	The actual list	Choice of character is between Human , Huntress, King, Samurai, Warrior, Wizard	PASS	Normal	9:48

I4	Human	Preview Image of ‘Human’ character is produced when the character is chosen	PASS	Normal	9:49
I5	Huntress	Preview Image of ‘Huntress’ character is produced when the character is chosen	PASS	Normal	9:50
I6	King	Preview Image of ‘King’ character is produced when the character is chosen	PASS	Normal	9:52
I7	Samurai	Preview Image of ‘Samurai’ character is produced when the character is chosen	PASS	Normal	9:54

I8	Warrior	Preview Image of ‘Warrior’ character is produced when the character is chosen	PASS	Normal	9:55
I9	Wizard	Preview Image of ‘Wizard’ character is produced when the character is chosen	PASS	Normal	9:57
I10	Start Button placement and structure	Placed at the bottom of the screen and blue in colour	PASS	Normal	(see proof)
I11	Back button placement and structure	Placed below start button and blue in colour	PASS	Normal	(see proof)

I12	Start button functionality	The start button then progresses our players onto the main game itself	PASS	Normal	10:10

The game itself

Test ID	**Test Subject**	**Expected Outcome**	**Pass/Fail**	**Data Type**	**Video Link/Proof**
J1	Display	A background image is produced from a random set of 3.	PASS	Normal	(see proof)
J2	Health Bar	Health bar for each player is displayed above their chosen character	PASS	Normal	10:13
J3	Score	P1 and P2 scores are displayed below health bar	PASS	Normal	10:13

J4	Countdown	Before the game starts a countdown is shown.	PASS	Normal	10:13
J5	Countdown Movement	Players cannot move until the countdown is finished	PASS	Normal	
J6	Idle animation	When not moving, characters perform an idle animation	PASS	Normal	10:14
J7	Auto Flip	Characters on the screen will automatically flip to face their opponent	PASS	Normal	10:35
J8	Screen boundary	Both characters cannot move out of the frame of the game or go above or below it	PASS	Boundary	

J9	W key	Jump animation for corresponding character plays and player 1 can make their character jump	PASS	Normal	10:27
J10	Gravity	After jumping the corresponding character should automatically come down	PASS	Normal	10:28
J11	Double jump	Players should not be able to double jump	PASS	Normal	
J12	A Key	Walking animation plays and player 1 can move their character backwards	PASS	Normal	10:27

J13	D Key	Walking animation plays and player 1 can move their character Forward	PASS	Normal	10:27
J14	R Key	Attack animation 1 plays for the specific Player 1 character. This is the low damage move hence will do moderate damage to the opponent	PASS	Normal	10:40
J15	T Key	Attack animation 2 plays for the specific Player 1 character. This is the low damage move hence will do high damage to the opponent	PASS	Normal	10:41

J16	Up arrow Key	Jump animation for corresponding character plays and player 2 can make their character jump	PASS	Normal	10:16
J17	Left arrow Key	Walking animation plays and player 2 can move their character backwards	PASS	Normal	10:16
J18	Right arrow key	Walking animation plays and player 2 can move his character Forwards	PASS	Normal	10:16

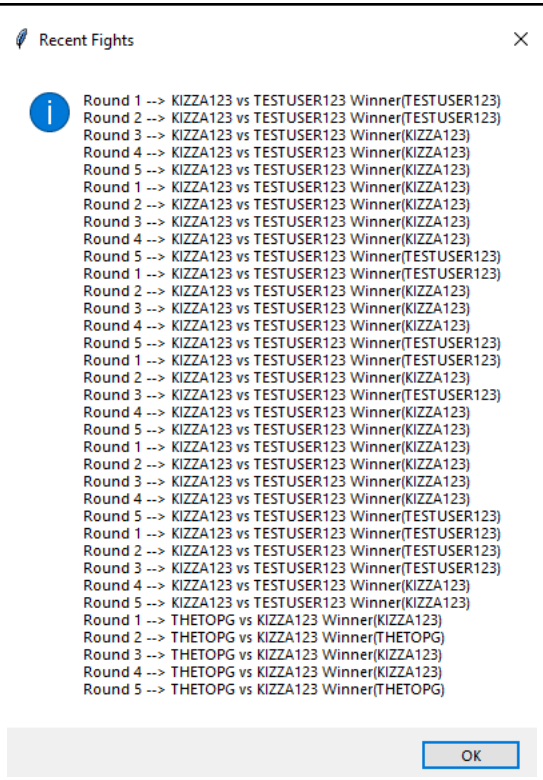
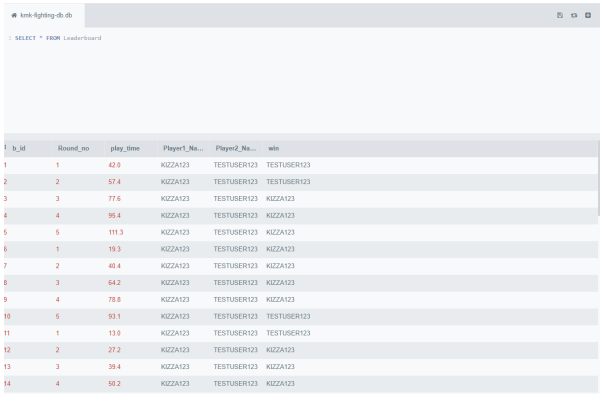
J19	Numpad 1	Attack animation 1 plays for the specific Player 2 character. This is the low damage move hence will do moderate damage to the opponent	PASS	Normal	10:22
J20	Numpad 2	Attack animation 2 plays for the specific Player 2 character. This is the low damage move hence will do high damage to the opponent	PASS	Normal	10:22
J21	Health bar (after being hit by an attack)	Each attack depletes the health bar	PASS	Normal	11:06

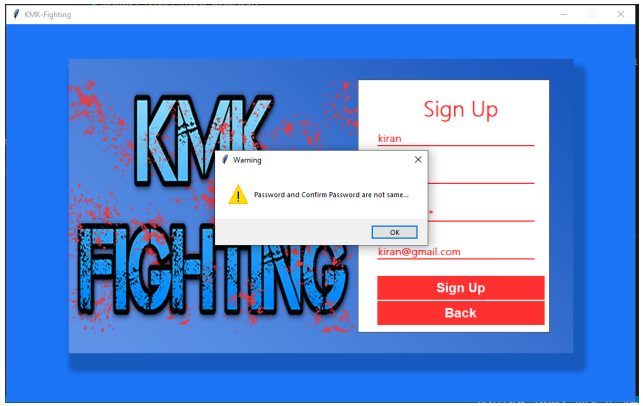
J22	Defeat animation	A round is won by depleting the opponent's health bar. Then an animation should play to show the defeat of the character that has just lost	PASS	Normal	11:15
J23	5 rounds	The entire game lasts 5 rounds	PASS	Normal	12:33
J24	Score	Each time a player depleted the opponent's health bar, the score is incremented at the top	PASS	Normal	11:14
J25	Victory Message	An image that says victory will pop up on the screen	PASS	Normal	11:14

J26	Data storage	All the data from the game such as round wins and average playtime are then stored to the external SQL table	PASS	Normal	(see proof)
J27	Game Music	Add my own game musics	FAIL	Normal	

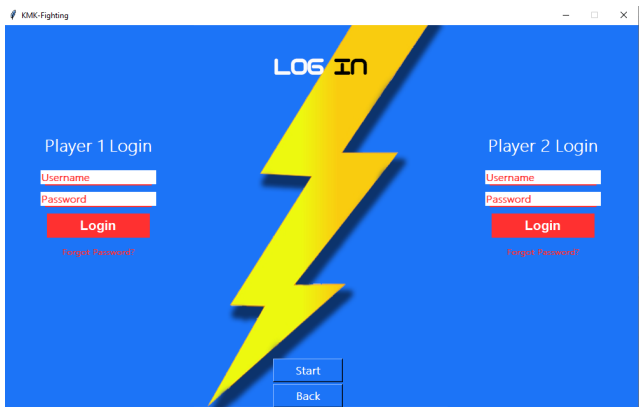
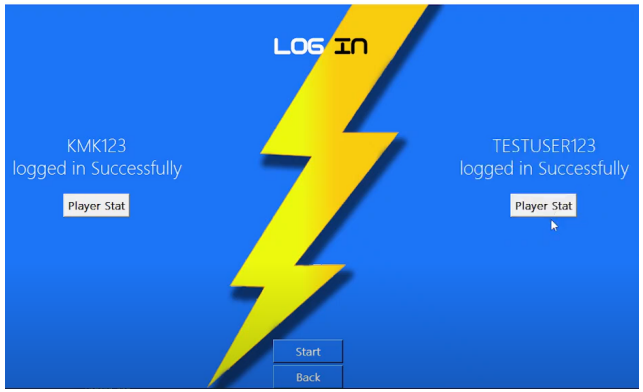
Test Proofs

Test ID	Proof	Further Comments
A1, A2, A3, A4, A5		It is different to my initial design which featured a central menu and a background but the functionality remains the same

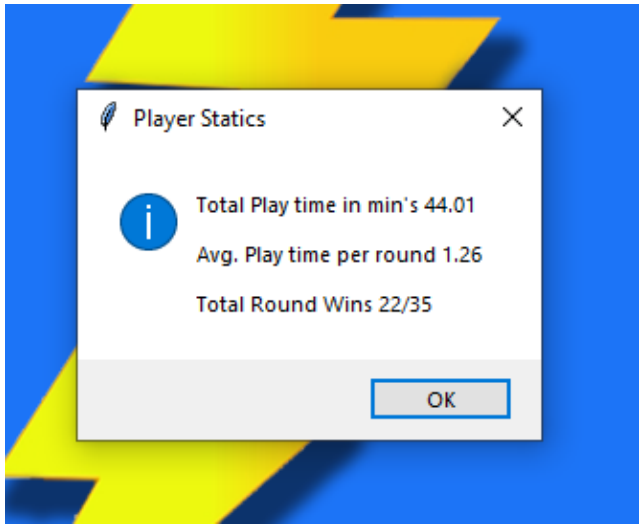
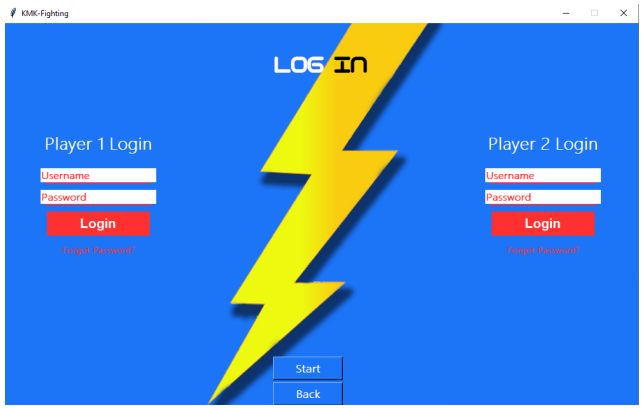
C2		
C3		First 15 fights

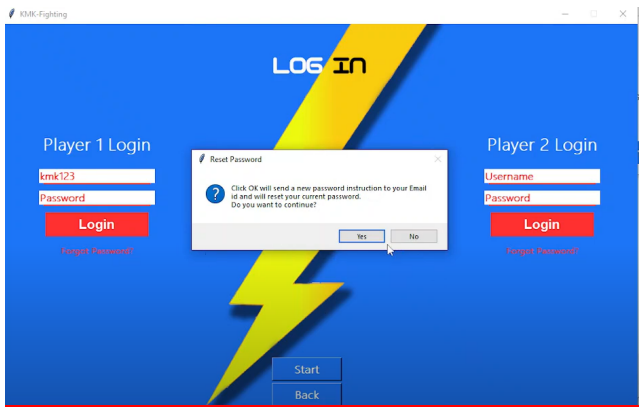
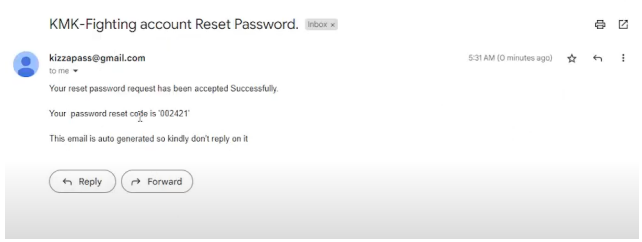
D2 , D3, D5		Initially was a blue colour however this colour was preferred as fields are clearer this way
D6		

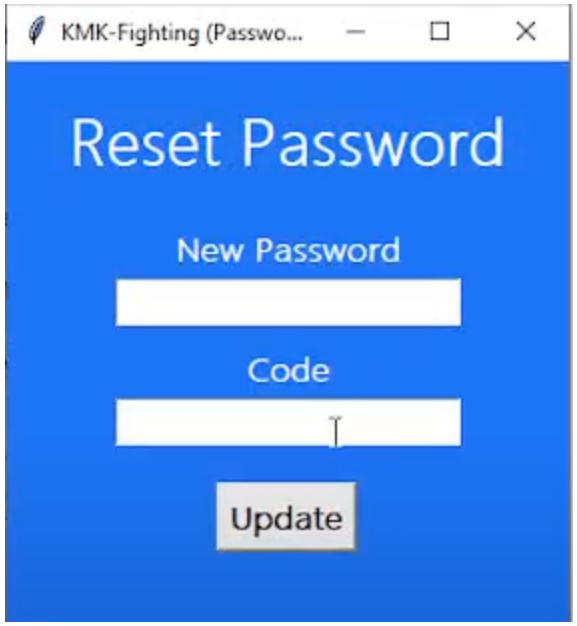
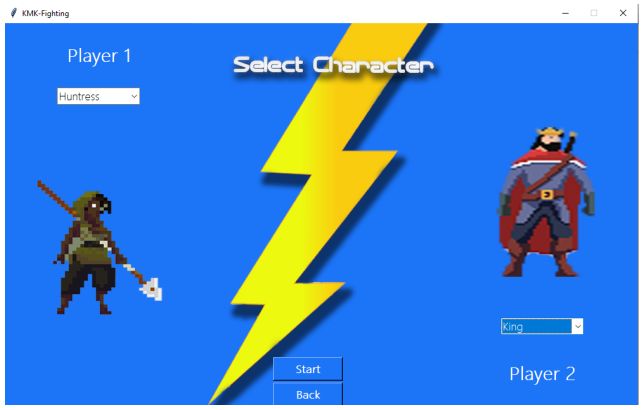
D7		Not exactly a user friendly error message but the code itself worked hence purpose is served
D9		

F1, F2, F4 , F5 , F6		The lighting bolt is a beautiful way to split the screen
F9		Stats for corresponding user
G3		Each round has a playtime and then my program will calculate an average for these and display it and store it under each user

b_id	Round_no	play_time	Player1_Na...	Player2_Na...	win
1	1	42.0	KIZZA123	TESTUSER123	TESTUSER123
2	2	57.4	KIZZA123	TESTUSER123	TESTUSER123
3	3	77.6	KIZZA123	TESTUSER123	KIZZA123
4	4	95.4	KIZZA123	TESTUSER123	KIZZA123
5	5	111.3	KIZZA123	TESTUSER123	KIZZA123

G4		Initially set to minutes however this is because there is a currency called game minutes which is externally being used by the gaming club. Hence to satisfy my clients it is in minutes but the actual program measures in seconds.
H1		

H2	 KMK-Fighting LOG IN Player 1 Login kmk123 Password Login Forgot Password? Reset Password Click OK will send a new password instruction to your Email id and will reset your current password. Do you want to continue? Yes No Player 2 Login Username Password Login Forgot Password? Start Back  KMK-Fighting account Reset Password. Inbox × kizzapass@gmail.com to me 5:31 AM (0 minutes ago) Your reset password request has been accepted Successfully. Your password reset code is '002421' This email is auto generated so kindly don't reply on it Reply Forward	

H4		
I1, I10, I11		Player 1 and Player 2 split again

J1		
J26		
J27		J27 was a fail as I could not import my own music to the game. However the game functions fine without it and was forced to use music assets from the web

b_id	Round_no	play_time	Player1_Na...	Player2_Na...	win
1	1	42.0	KIZZA123	TESTUSER123	TESTUSER123
2	2	57.4	KIZZA123	TESTUSER123	TESTUSER123
3	3	77.6	KIZZA123	TESTUSER123	KIZZA123
4	4	95.4	KIZZA123	TESTUSER123	KIZZA123
5	5	111.3	KIZZA123	TESTUSER123	KIZZA123

EVALUATION

MENU SCREEN

SUCCESS CRITERIA	COMMENTS
❖ User is presented with a menu screen with a background image loaded up. A user interface is presented to the user with buttons for navigation: Play, Sign Up, Recent Fights, and Exit button	Fairly simple to implement however I did have to ensure a box was made on the image so the buttons can be shown on the screen. Furthermore dimensions and assets had to be well organised and carefully fit the dimensions and resolutions of background images to ensure good graphics The buttons are built using the Button function and dimensions are defined within

❖ The Recent Fights button will show users the winners of every round in the past fights that have occurred. This button will open a window to show this. Each round is displayed along with players involved in the fight. And finally, the winner of the round should also be displayed. This will help me to store round wins to their corresponding data	In terms of functionality and purpose this hits every aspect. However, personally I believe the interface could have been much better, suiting more to my initial design of it. However as stated above it serves its purpose and is helpful for the many calculations my program does
❖ New players will be able to create an account via sign up button. A user must create an account before playing the game or else they will not be able to play the game.	A signup and login page is the mainstay of this whole program and using SQLLite and more button functions I was able to generate this for my client so that members of the club can store their fighting stats
❖ To begin the game. Players will press the play button which should bring them onto the login page where they can enter their credentials and can view their stats	Easy to implement and the image was well placed by me to split the window in half. Same boxes and fields were used from the login page in terms of colours.

Sign Up Page

SUCCESS CRITERIA	COMMENTS
❖ Should display fields for the new user to fill in	Use of <focusin> functions and creating frames allowed me to carry out this task fairly simply with clear boxes for users to enter.
❖ Basic credentials must be taken such as username, password and email	Boxes are well sized and the text is maintained within the constraints of the field. Font size also fits well. Overall a good job and simple to carry out
❖ A check must be carried out to ensure that all fields are filled out	Done fairly simply by using an else statement and works just fine
❖ An additional check must also be carried out to ensure the user's passwords match	Compare both passwords, else statement, done sorted
❖ Yet another check should be carried out to ensure that the email address is not already registered	For this one I did not manage to generate my own error message however SQLite provides me with its own error message and the actual check is carried out in SQL rather than python hence why error message is a little technical
❖ Final check should be carried out to check the syntax of the email to see if it is valid	And finally this is done in the main python code by specifying the structure of an email and comparing the user input to it

Login Page

SUCCESS CRITERIA	COMMENTS
❖ After the play button is pressed. A login page should be presented for each of the players. Valid details will let them successfully log in and play the game	Background image is produced and fields for both players are given. A forgot password button is placed below the user fields
❖ In addition, a 'forgot password' option should be presented. Users can use it to reset their password via email	See above
❖ In the event that a user chooses to reset their password. The user should be presented with an external window. This external window will allow users to enter their details and the corresponding email will receive a code which they can use to reset their password	This was implemented fairly successfully in my opinion. Does what it says on the success criteria. Using SMTP and Encryption protocols I was able to send emails from my email to a corresponding user's email. Code is received and users can update their password (given the code is right) and this is updated on the users SQL table
❖ After a successful login, an option should be presented to each user where they can view their stats such as total playtime, total wins etc. This should be presented in an external window	This is where the aggregate SQL comes in. The values are fetched, calculated and displayed every time rather than values being stored in more tables. This allows members of the club to compare stats with each other

❖ If both players are logged in and the start button is pressed. The Character Select screen should be displayed for them	Implemented, See below for comments on character select

Character Selection

SUCCESS CRITERIA	COMMENTS
❖ Both players should progress to a window to choose their character for fighting	The screen itself was easy to implement, same background as before
❖ This will be a dropdown menu for each user. They can choose from a total of 5 different characters. Each character should have different damage properties.	In my opinion this was not simple to implement. The file directory of the assets were almost like a list and this list was forward by one which means all the characters previewed were wrong. This was fixed by adding an empty slot into the list and then pushing the character's to their corresponding name. This ensured that when a character was selected the correct image was displayed
❖ A preview of the character should be shown before each user confirms their character	Image generation itself was no issue but as stated above that issues arised from disorganised assets which was fixed thankfully
❖ When both users have selected their character then the game can be started	If statement does the job just fine

The game itself

SUCCESS CRITERIA	COMMENTS
❖ The orientation/layout should begin with player 1's character on the left and player 2's on the right	Simple to implement. Just had to ensure that Player 1's character was the one on the left and vice versa as issue mainly arised from swapped start position. But it was fixed and the x y coordinates are all correct now
❖ Health bars for each player should be displayed at the top along with the score counter below it	This was done by drawing green rectangles to represent the health bar
❖ Player characters should be able to move in any 2D direction and should always face the opponent character	Done fairly simply after understanding the animation as just an iteration through multiple frames. Variable such as y velocity were used to change coordinates of characters when buttons were pressed

❖ Each character should have different attacks. Which can be used against the opponent to deplete their health bar. The damage dealt should be designed to depend on the character itself and what attack they are using	Damage dealt is nothing but a red rectangle over the green rectangle (health). The size of the rectangle increases as more collisions occur and they depend on the damage the opponent's character does. Furthermore there is a variable that is decreasing every time an attack hits. Different attacks is done through iterating through a different part of the sprite sheet based on what type of attack is being used
❖ The first user wins a round by depleting the opponent's health bar	As stated above, the variable is there to determine hit points and when it reaches 0 the opposing player wins the round and they have 1
❖ The whole game should be 5 rounds in total. The player to win the most rounds will win the entire game	Done fairly simply

Client Feedback:

What do you think about the graphical side of the game? Please tell me what you like and don't like

I think the graphical user interface and the characters, backgrounds etc are all quite pleasing and is definitely something that could catch my eye. The backgrounds are stellar, the characters and their animations are quite unique. In terms of improvements I would say to fix the warrior character in the game as he can be shown a little blurry on the actual fight scene and the recent fights window could be more pleasing to look at

I agree with my client as the recent fights window and the warrior character are not very good to look at due to sprite sheets and sizing issues which is definitely an area for improvement. However the rest of the game looks crisp, sharp and pleasing

In terms of user friendliness, how is my program? Is it something that is easy to navigate and understand? Any further improvements/ suggestions?

The program is very user friendly and each button, field and the overall navigation of the system is well laid out and concise. In addition the game follows a simple and understandable sequence to the combat arena itself. The login instructions are clear and easy to follow and users can sign up easily too. One thing I would like to suggest is that there was no need for the back buttons as players couldn't exactly un login if you know what I mean

I think I understand what my client means. When both players login and progress to the character selection screen they still have the option to go back to the login screen. My user is asking me what the point of that is if players are already logged in. While yes I agree with my client, the back button is actually there if the users want to check their stats. But other than that I think that giving an option to re-login by going back is definitely an improvement

After 1 week of the game being introduced to the club, how is it?

I think the game has been a huge success with the members already. It is very engaging and competitive. All the club members have to do is bring in their version of the game and it works on all the school PCs and we all are starting to have a good time even playing it during lessons. I have arranged a tournament for the game with some big prizes so I think the game will be an even bigger hit

While I don't particularly condone my game being played during lessons it is good to see that the gaming club is enjoying it and they are getting more people to come

And finally, in our first ever interview you said you would rather play your own console rather than an exclusive game for the club. Do you still feel the same way?

No, as now that this game has introduced the sense of community , competition and brotherhood this club once had is slowly improving and it is a nice feeling. Furthermore having a game you can play anywhere in the school is great

Again I don’t particularly condone my game being used outside the gaming club, I am glad that my program has been a success and changed the trajectory of this once great club