

OCR A LEVEL COMPUTER SCIENCE NEA

NAME: SAAD AKRAM

CANDIDATE NUMBER: 7166

CENTRE NUMBER: 10152

Written by Saad Akram

Contents

Analysis 3

Design 26

My Development Diary (Coding the Proposed Solution) 55

Evaluation 115

Appendix 130

Analysis

Brief Introduction to Problem

In 1987, an iconic arcade game called “Street Fighter” was released by the Japanese developer Capcom. It was a massive success in arcades and set the standards for the fighting games until today – this includes special attacks, six-button controls, and the use of command-based special moves.

In the game, there are two modes – single player, where the player competes against the CPU – and 2-player, where you compete against another real player. The aim of the game is to win 2 rounds out of 3 – to win a round, the player must deplete the opponent’s energy to 0 within 30 seconds (knockout). However, if neither player manages to knockout each other, whoever has the highest level of health by the end wins the round.

My intentions with this project, are to create a game like Street Fighter and incorporate some new features to give the game another sense of objective. Also, I would like my game to have more smooth, modern animations to be more appealing. Notwithstanding, I will ensure to keep the core elements of the original game.

To achieve this, I will analyse the original game in a great amount of depth. To better gauge what features I should add or ignore, I will discuss my ideas and objectives with a suitably identified stakeholder. Moreover, I will look at some gaming forums discussing the original game and what features they would like to see in a spin-off of the game.

Identifying my Stakeholders

Who are the suitable stakeholders?

My target audience for this game is from the ages of 7 to 15. While the lower end of this range may seem too young for a fighting game, “Combat Warrior” is a fighting game between two animated fantasy characters which does not depict realistic injury or trauma. Instead, health bars are used to represent this, appearing more as a score rather than genuine violence. It is intentioned to be a game that displays the artistic combative styles of the fantasy warriors that are not comparable to real life. Also, due to the characters being animated, it means that the visual style is not overly realistic hence making it appropriate to slightly younger audiences. The game will be designed to play on PC or laptop, so it is very accessible for most kids as they can use them in libraries, at home etc. Furthermore, the gameplay controls and mechanics are simplistic and easy to understand, making it suitable for a younger audience.

The game is overall meant to allow for competition, enjoyment of fantasy animations and designs, as well as an appreciation of the simplicity of this “Street Fighter” style game.

From this criterion, I have been able to identify an appropriate stakeholder in my project. Mateen Raza, who is a 12-year-old Middle school student living in Denmark. He is deeply interested in games and is very involved with action and combat games such as “Brawlhalla” and “Tekken”. He also has his own gaming PC (although my game will be compatible with

all) so he will be the perfect candidate for testing the gameplay mechanics and features – this will be helpful for me as I can receive constant feedback throughout the process to further guide the direction of my project.

From my discussion with him, he has explained that a fighting game with a retro feel, but modern graphics and animation would suit him best. He feels that my focus on competition and adding a fantastical element to the game would bring him enjoyment and most importantly, quality time spent with those who he plays with. Moreover, he explained that he felt modern 1 on 1 fighting games were overly complex and take away from the core fundamental aspect of this type of game. This further justifies my decision to make the controls less complex though not lacking depth or scope. Another aspect which Mateen mentioned was he felt most new fighting games were too graphic and realistic, which can be disturbing due to his age.

It is clear that Mateen's needs align with my vision for the game and that he would be suitable in guiding me towards a game which satisfies the relevant target audience.

Research of the problem

Breaking down the Original "Street Fighter" Game

The core fundamental aspect of the game "Street Fighter" is two players fighting in a 3-round match. A player has the option of one of two game modes: "ONE PLAYER" or "PLAYER1 VS PLAYER2" (one player or 2-player mode). This selection is done through the menu screen and the 2-player game mode is only accessible through putting in 2 or more credits.

In the one player game mode, it is only possible to advance to the next opponent once the current opponent has been defeated and there is only one playable character if this mode is selected from the get-go – "Ryu". However, if the player advances to the next match and loses, he has the option to replay instead of resetting the streak. This option is given after losing and if not selected, the message "Game Over" is displayed.

On the other hand, if two-player mode is selected, the winning player goes on to face the rest of the opponents in one player mode. So, it is possible to not use "Ryu".

Once the mode has been selected, two options are displayed on the screen – Japan or USA (represented by the flags) – so the user can choose where to fight. Based on this, the selection of opponents to fight consecutively is different. However, in both settings, the main character "Ryu" is the playable character unless the opposing fighter in two-player mode wins.

The fighters are displayed before the first round begins. To win a round, the player must reduce the other players energy down to 0. It is best 2 out of 3 rounds. If the player loses the match, he has the option to continue– a cutscene of a ticking bomb appears which gives the player 10 seconds to choose.

If the player does continue, he may rematch the fighter who he lost against until he beats him. If not, the player is redirected back to menu screen where he can select game modes.

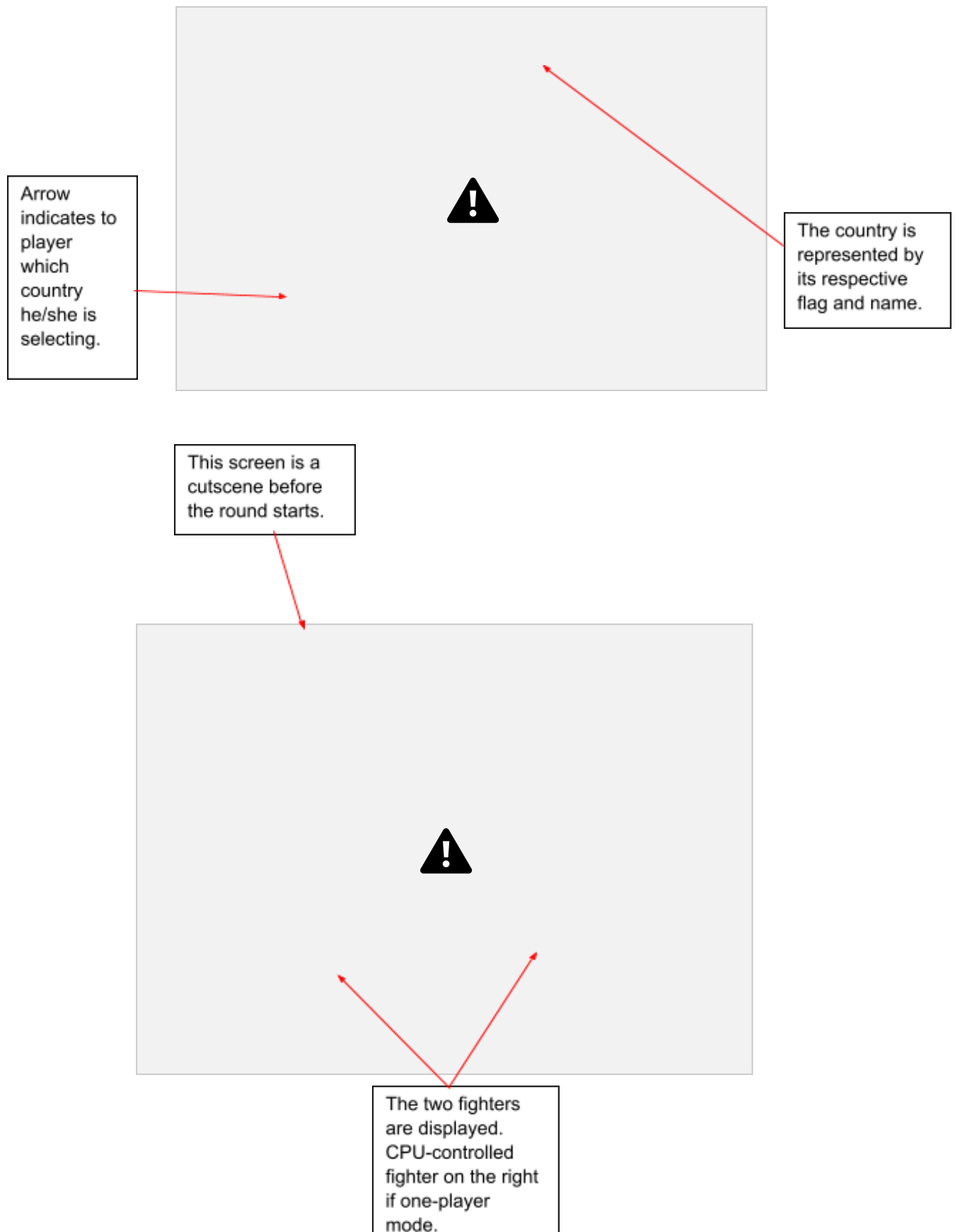
As the player continues to defeat players, he will be able to access parts of the map hidden at the beginning.

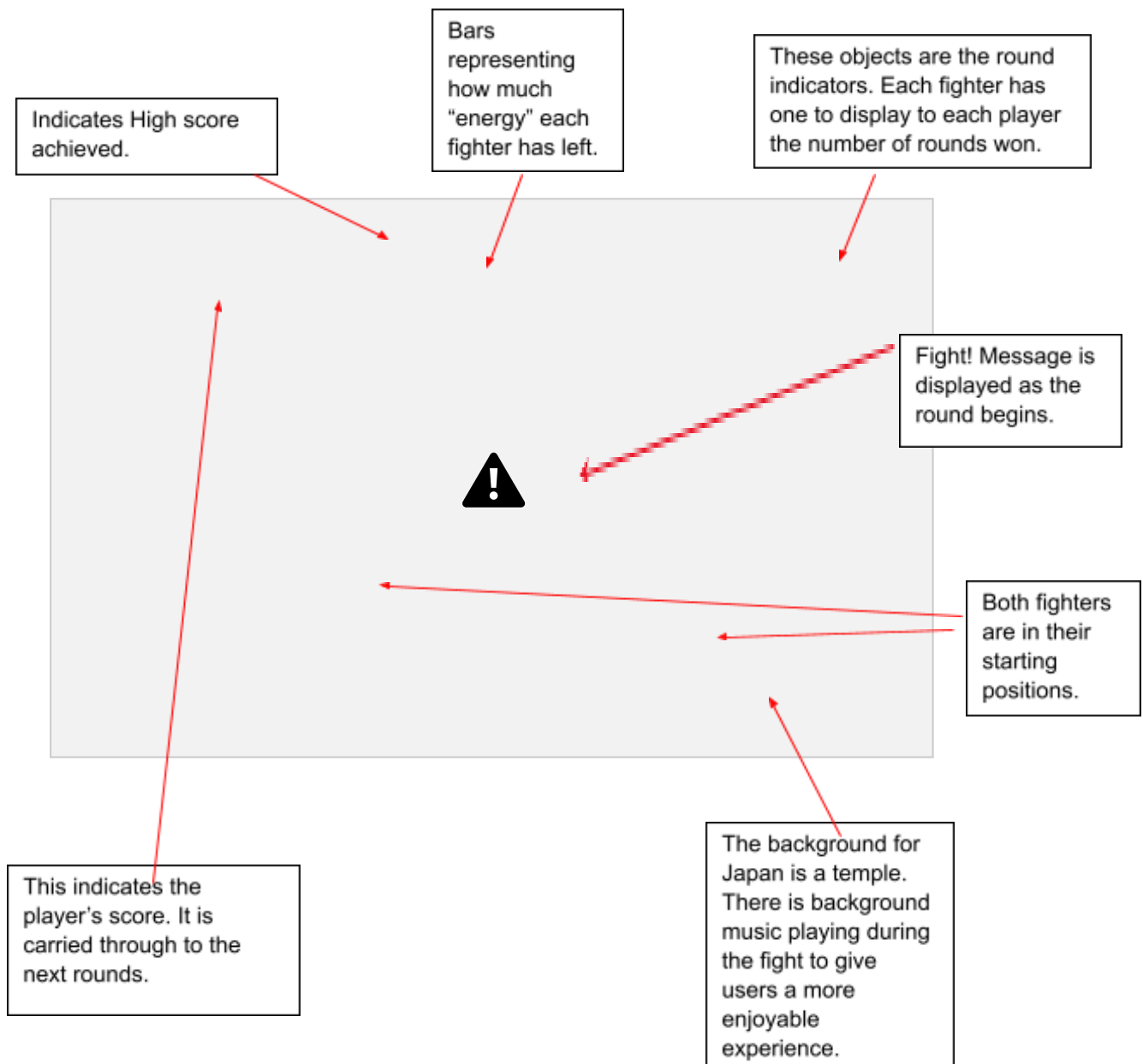
Street Fighter does not have a traditional end point or conclusion, instead after defeating all opponents in single player mode, the game loops back to the beginning. However, in the second iteration the difficulty is increased. This keeps occurring until the player decides to end the game or runs out of credits.

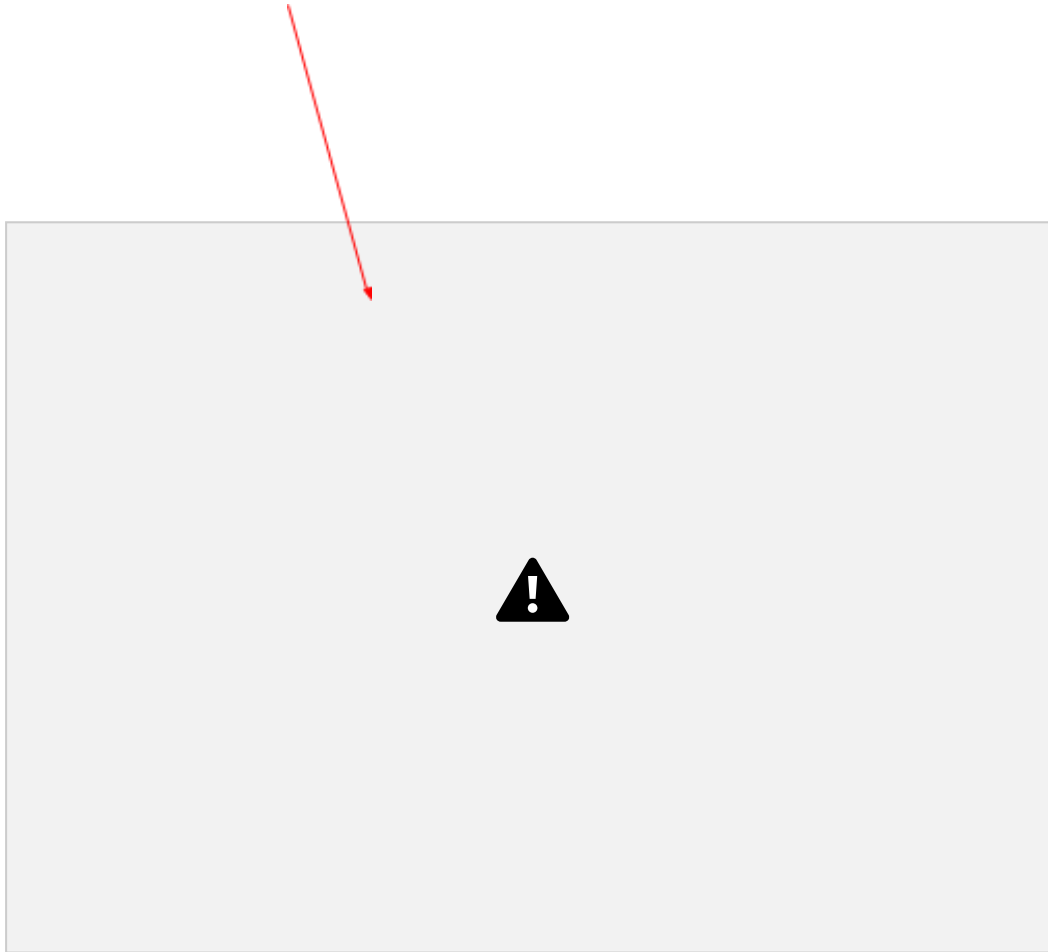
Overall, Street fighter 1 was more focused on the competitive element of one-to-one combat rather a story-mode style game. As a result, the goal is to achieve higher scores by defeating as many opponents as possible. Below I have some illustrations of analysing different key stages and mechanics of the game.

The Menu Screen

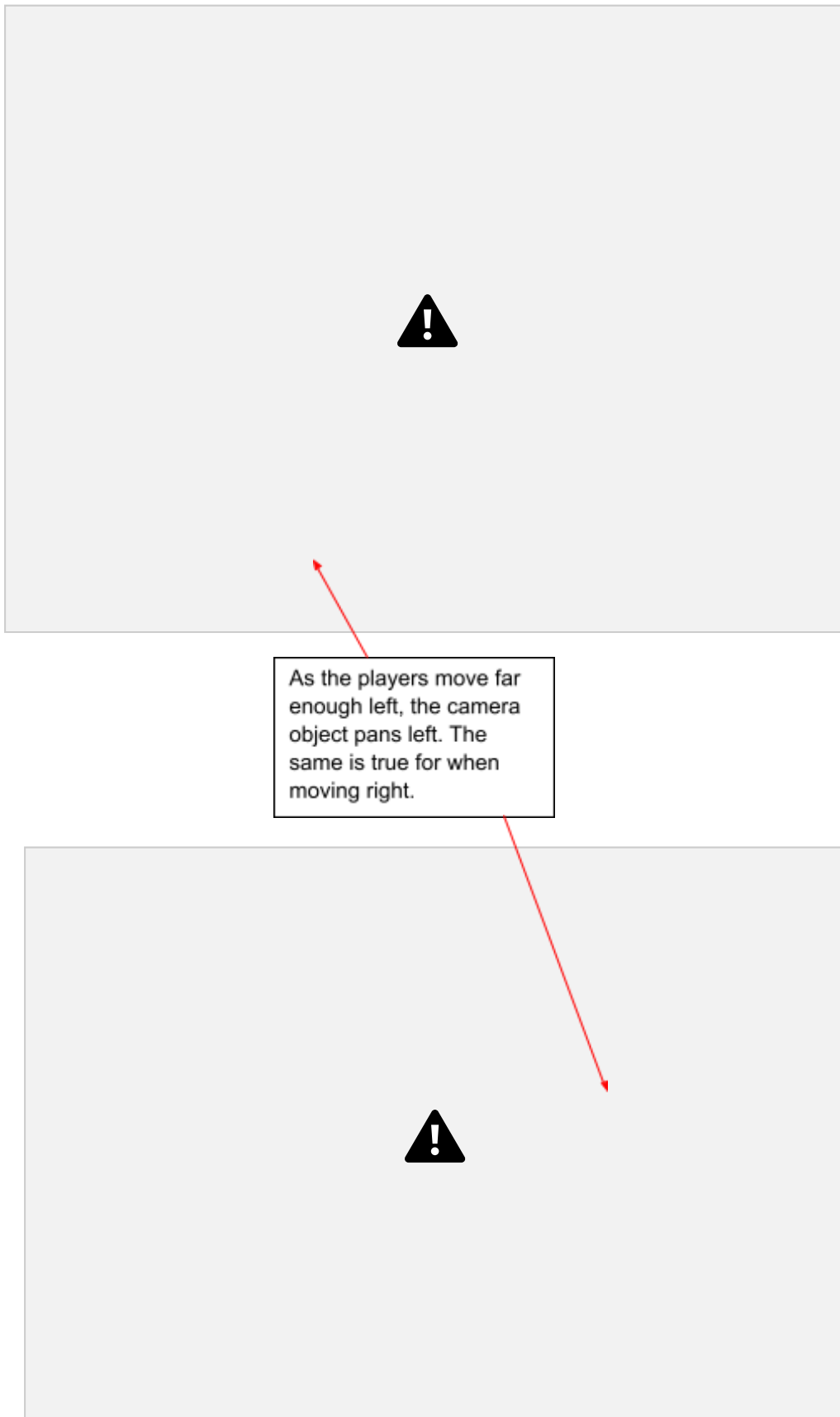


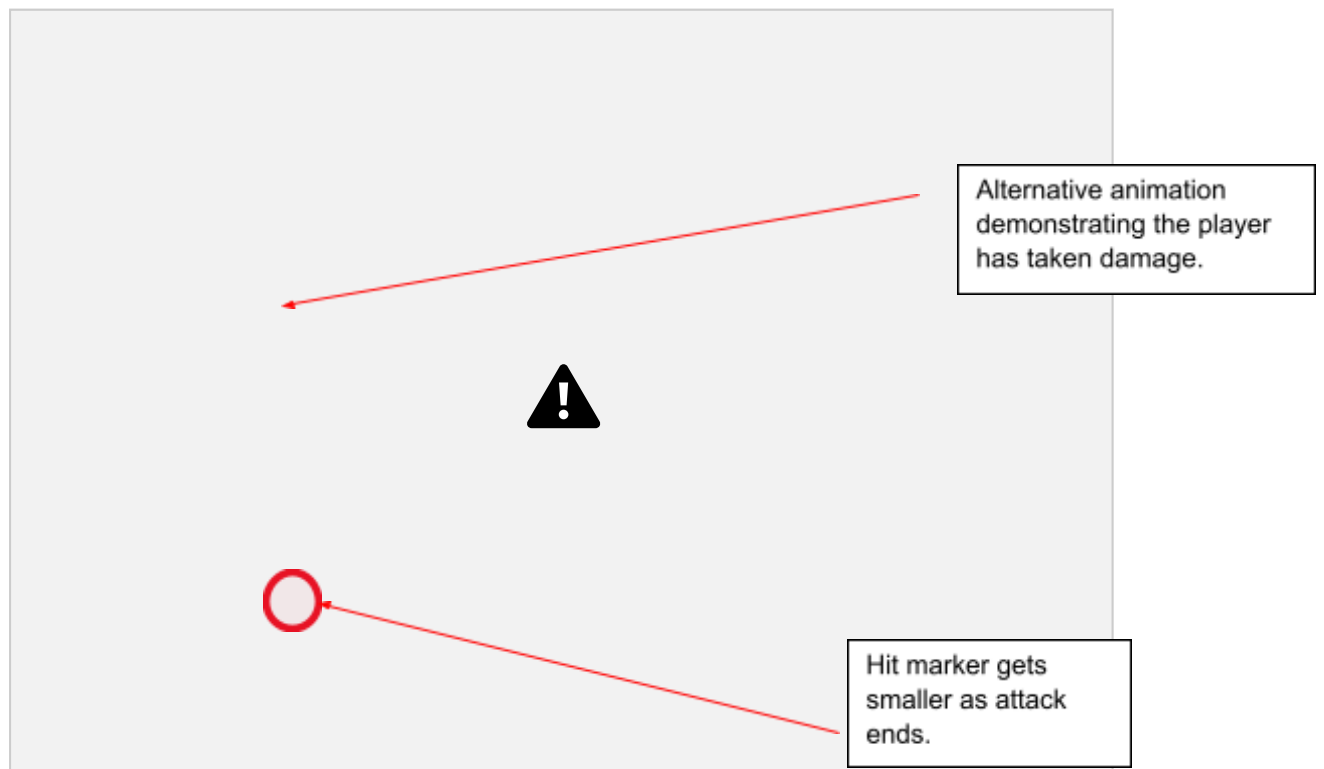
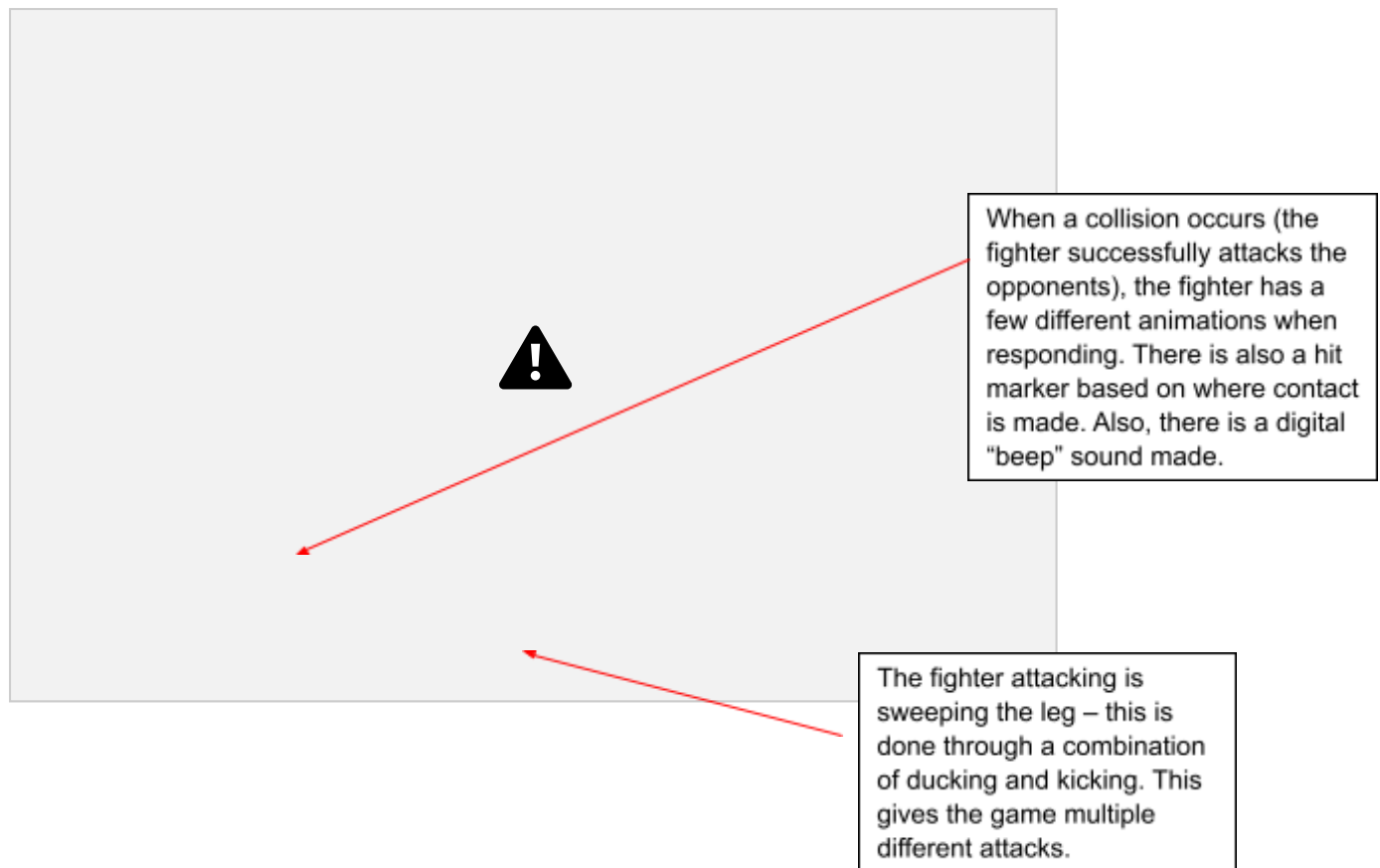
Country Select + Round Start cutscene

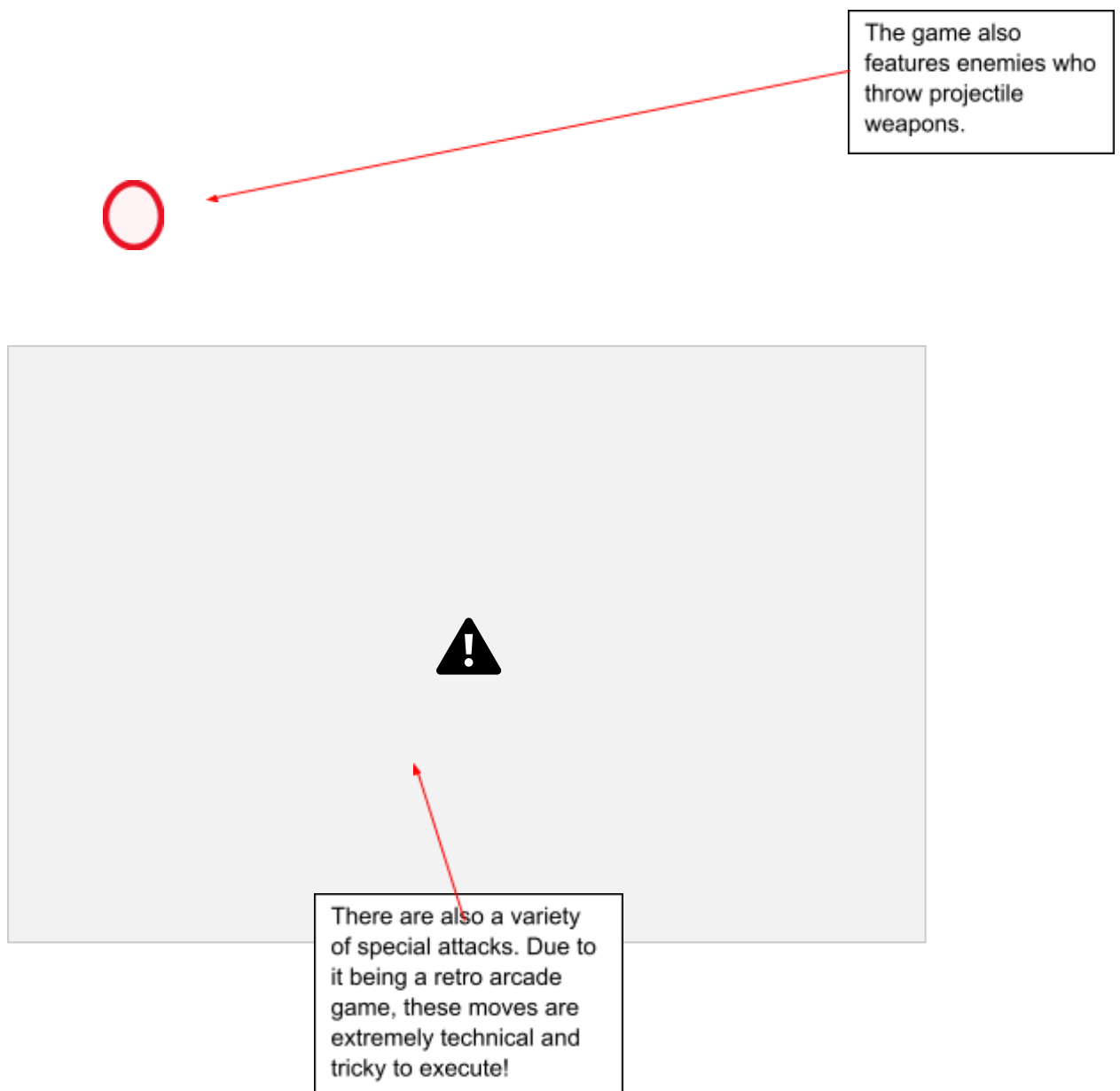
1st Round Start USA and JAPAN



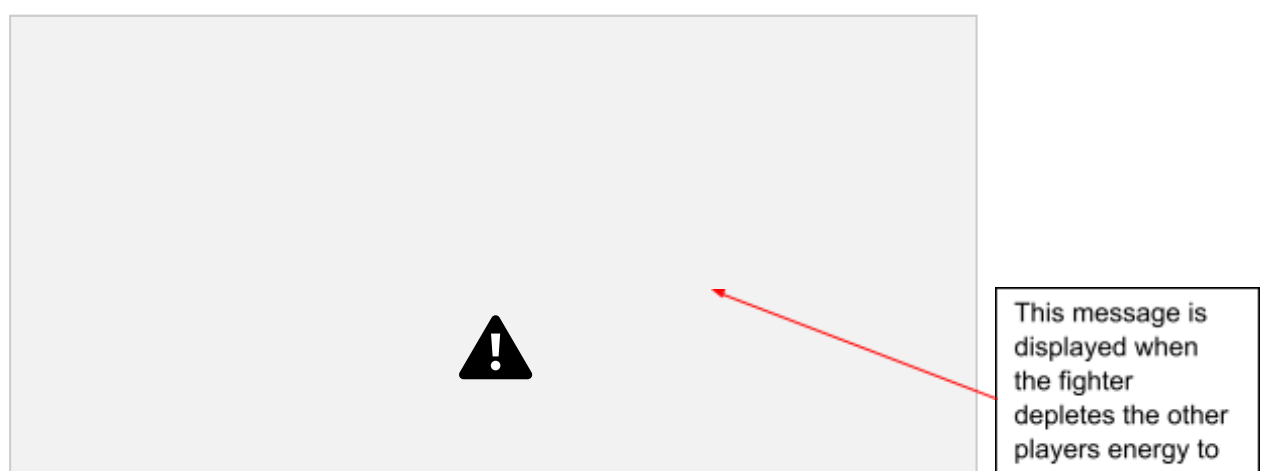
Camera View (L+R)

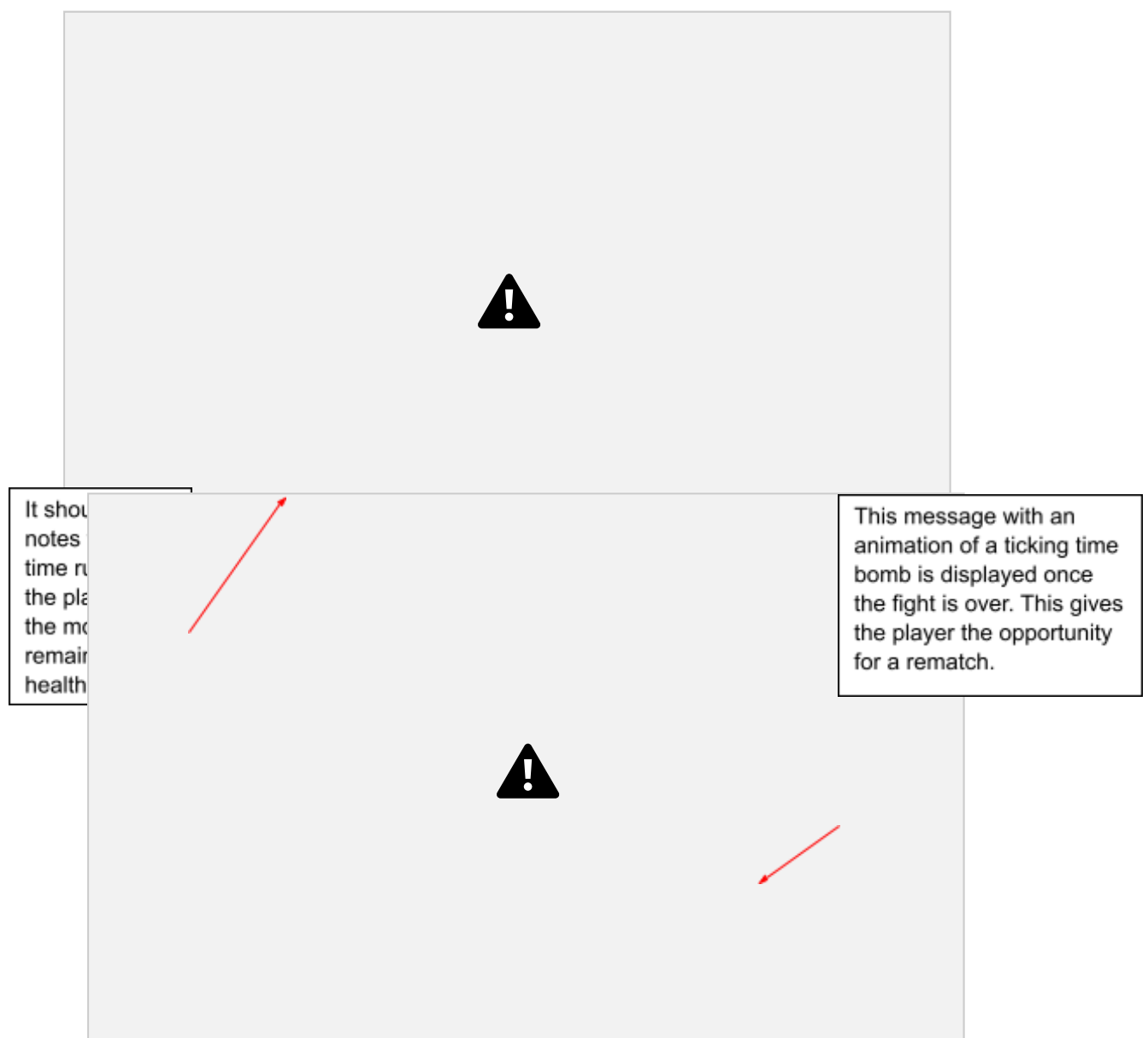
**Attack Animations (Mid Round)**





Cutscenes and Transitions





Once the time bomb runs out, the bomb animatedly explodes, and the following message is displayed. This causes the game state to transition back to the main menu where the player may restart the entire process.



Key Stages





There are also 4 bonus games. These occur every 2 stages (i.e. every 2 bosses defeated). These bonus games allow the player to add to their high score.

Summary of key points of interview with Stakeholder

I chose to interview my stakeholder, Mateen Raza, to discuss the features of the original game which I decomposed and decide which bits of the game he would like to see implemented or forgone.

I began by asking him whether he thought the menu screen should emulate the arcade by using an "insert credits" system and if he wanted my game to have a retro-feel. He addressed this and did not feel that his demographic would enjoy a retro-style game. Instead, he would like it to be a more modern interpretation of "Street fighter". Moreover, he added that he would like there to be a character select on the main menu and this would give the player access to different fighter and styles as he felt the original game was too one-dimensional. This aligns with my vision of creating a game which is artistic and where a multitude of styles can be experienced.

Next, I inquired about the two game modes, One-player, and two-player mode, and which he thought was more essential to the game he would like to see built. Overall, he felt that he would like to see both game modes included in my game. However, when I asked a follow up question, he said that he would not like it to be a story-mode game. He stated that he would much rather the simplicity and control of being able to select the fighters and battle. This according to him, would make it easier for the user to find interesting matchups without having to struggle through many complicated stages of the game.

I then queried him about what metric would make the game competitive with other players in one-player mode. His response was that he would like to have a scoring system which saved the high scores achieved. This would require the implementation of a database link where a name would be entered every time a user played the game to store the scores.

Subsequently, I asked him about any other features of the scoring system he would like to see added to which he expressed an interest in including a stamina bar along with the health bar. He further added that in conjunction with the fighter depleting each other's health, that each time an attack is initiated, the fighter would lose stamina, and this would be indicated by a separate bar to add a layer of complexity and realism to the fight. By losing stamina, the player would be restricted in the types of attacks he can perform if it drops below a certain level or perhaps lose more health when attacked. In addition, when questioned, he said that he would like to see the timer retained in my version of the game but would want slightly longer rounds.

Afterward, I asked about the attack mechanics and what he would like to see implemented in my game. He mentioned that he would like to have a more simplistic attacking arsenal, but he wanted the attacks to be more fluid and artistic than the original game. Moreover, when inquired about special attacks, he conveyed to me that he wouldn't like them to be implemented in the game. His reasoning was that if someone were to figure out how to consistently do these attacks, it would become unfair and very unbalanced. He would prefer to just have simple attacks which the player would have to skilfully use to take his opponent out rather than "spamming" combinations.

Also, I asked him about the bonus game feature the original "Street Fighter" included. He expressed to me that this would be a cool feature to have separate to fight mode. However, he said that it was not majorly important and would rather more focus be put on improving the fight mechanics and features.

Finally, a key point he made when asked, was about being able to select the maps which were fought on. He stated this feature is included in a popular game known as “Brawlhalla” where there are several fantastical and grand backgrounds where the battles would take place.

To conclude, overall Mateen’s input was very helpful for me to assess the features to include in my game “Combat Warrior”. Apparently, he wanted the game to not have the stage-by-stage style included in the original game. Instead, he felt that giving the user control of player and map selection would make the game more interesting and dimensional. It was useful to see where my game can differ to suit my stakeholder demographic.

Essential features of the proposed solution

In this section, I will clearly identify and justify the features to be included in my end-product. This will be based on my interview with my stakeholder, as well as considering my vision for the game.

Identified features	Justification/Explanation
Various selection of playable warriors	This allows the player to have more control and selection in the game which ultimately gives the game a customisable feel. Also, in my discussion with Mateen, we decided giving the player access to different characters with unique styles would be far more exciting and varied.
Modern menu screen	Mainly stemming from my interview with Mateen, it was clear that a menu screen with buttons for “Play”, “View Leaderboard” and “Exit” would be more intuitive and modern than the credit system emulator included in online interpretations of the original game.
Unique fighting styles for each warrior	To give the player a chance to experience a variety of animations which emphasises my subtle artistic vision for the game. Moreover, it adds depth to the gameplay, as players will have to adjust the way they fight according to the characters style, range, and attack selection.
Clearly defined attributes for each character	In order to create interesting matchups between the warriors. This adds to the uniqueness of each character. It adds a tactical component to the game. As a warrior with greater higher defence may be played in a more passive or patient manner than one specialising in attack.
Modern, smooth 2D game graphics	As discussed with Mateen, this adds a modern edge and feel to this retro-inspired game. More fluidity in the gameplay is also more captivating to a younger audience which is my target demographic.
Smooth character animations	Makes the fighting mechanics of the game more pleasing to view and clear to the player. This also

	aligns with Mateen's view of how the game mechanics should feel.
Intuitive and simple controls for warrior move set	Due to this making the game easier to learn as well as suiting a wider range of age audiences. This was also discussed with Mateen, who wanted controls which felt more intuitive and not so complex in nature.
Combinations between movement and attacking controls	This adds to the game intuitive nature. Instead of different buttons being used to perform one attack in multiple ways (such as a button for high attack and low attack), using a combination of a movement button and attack button to form a low attack would make a lot more sense and allow for greater ease of play. This was affirmed by Mateen who stated this is a more modern approach to the controls.
Visible health bars to indicate state of each warrior	Removes unnecessary effects and realism in the game. Makes it much easier for each player to track how much health they have left. This is important as the player may adjust his fighting style accordingly.
Stamina bars to indicate energy of each fighter	As suggested by Mateen, this is done to add another dimension to the scoring of the game. Instead of just health bars, player will have to monitor their stamina, and this makes the game more tactically orientated.
Multitude of playable maps	Adds more customisability to the gameplay and makes it more visually appealing.
Engaging Background music	Including tracks associated with martial arts and fighting gives the players a more enjoyable experience due to adding tension and depth.
Sound effects upon successful attacks	To give an indication to the player the nature of his attack. Different sound effects will be implemented depending on if the warrior lands successfully and cleanly, or is blocked, or completely misses its target.
Round indicators	So that players can view how many rounds have been won, and how many are left to win the game.
Fantastical warrior animations and skillsets	As discussed with Mateen, this makes the game feel more creative and awe-inspiring instead of standard boring characters. It appeals more to imagination which makes the game more intriguing.
Timer for Rounds	To restrict the players in terms of time. Adds pressure and exhilaration as there is more than just depletion of the opponent's health bar.

Limitations of Solution

Below I have identified any limitations in creating my game "Combat Warrior". I have also explained why these limitations may exist to provide clarity.

Limitation	Explanation and Justification
Two-player mode only.	This is a limitation because it means that I will not implement other game modes such as one-player only as discussed with Mateen. This exists due to not having enough coding knowledge or time to implement such a feature to a high level.
No multiplayer mode	This limits the range of players that the user can compete with hence reducing the level of competition available in the game. However, this is due to me not yet being able to implement multithreading and client-server communication as this is real-time gameplay.
No special attacks to be included	This limits how the set of moves a warrior may have available to them and may reduce dimensions of the fighting aspect of the game. This limitation exists as it would be too time-consuming and intensive to alter the sprite sheets which I will use for each character. It also suits the game well as Mateen believes this will cause players to fight only aiming for special moves (which could lead to spamming).
Graphics implementation	It is important to have clear and modern graphics in this game. However, I will not be able to implement advanced graphics such as shadow casting or dynamics lighting as this requires advance software and is very resource intensive. This not possible for me as I do not have access to these things.
Gameplay limited to only one keyboard	This means the controls for each character will be on the same keyboard in two player mode. This means additional hardware devices will not be compatible with my game as I do not have the resources to develop and test these.

How is the problem solvable by computational methods?

Why is a computer program suitable for solving this problem?

Considering that the game "Street Fighter" was originally produced as an arcade game and later consoles, programming is an integral part of the development process. This is due to programming allowing for code to be created which fundamentally responds to a user's input producing a specific output allowing them to control the warriors, move through the game setting and engage with other entities on screen. This type of interaction is not possible in

real-world situations where one may observe the fighters but have no influence on the outcome. Another reason is that programming is most suitable, is that it allows for a digital representation of different artistic combative styles and movement which may not be possible when engaging in real-world combat for instance. Therefore, using computational means are best suited to making the game more appealing and interesting to players.

Features that make the problem solvable by computational methods

I have listed below, some examples to demonstrate how certain features are suited to be solved by a computer program due to computational methods.

Thinking Abstractly

It is imperative that I use abstraction to decide what is necessary to include/exclude in my game.

Below I have listed the ways how I will implement it into my program.

Uses of Abstraction in “Combat Warrior”:

- Removal of unnecessary background sounds that are not relevant to the core elements of the fighting. This would include chirping from birds, wind noise or people who may walk past.
- Removal of highly detailed sound effects. My game will use simplified sound effects to indicate whether an attack has landed or when a round is over for example.
- Adding health bars on the screen to indicate to a player how much health a warrior has left instead of relying on realistic signs such as blood and injury.
- Using 2D graphics (retro style graphics) instead of 3D as well as having simple colour schemes so only the necessary details are used.
- Ignoring unnecessary details like shadows and weather.

Thinking Ahead

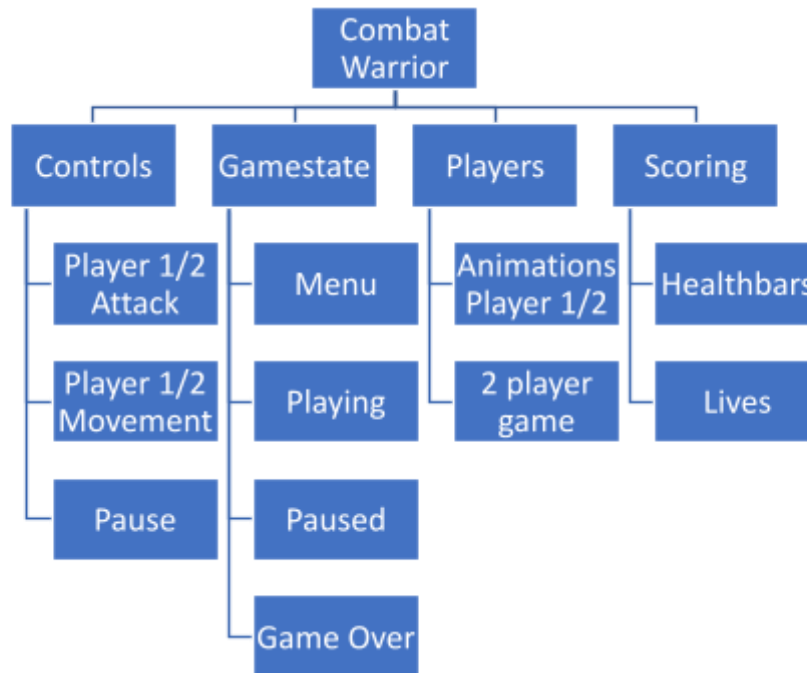
Thinking ahead refers to identifying relevant inputs and outputs to a solution before solving the problem. It allows for you to identify any potential problems, and these can be addressed to allow for enough time to solve these.

- I will use the Python library “Pygame” as this language is versatile and my existing skills in python can be used to create these games relatively easily. It also uses very convenient abstractions for handling things like sprites and events. This means I can focus on the program’s logic and flow more.
- The keys W, A, S, D will be used as inputs to control the characters movement on the left side of the screen and the keys Up, Down, Left and Right will control movement for the character on the right side of the screen.
- Certain sound effects will be outputted to indicate certain objectives met by the player such as a successful attack or movement across the screen. Also, the opposing players health bar will deplete if the attack was successful.
- Incorrect presses of keys will have no relevant output to hint that the wrong key was pressed.

Decomposition of the problem

Decomposition refers to me breaking down a problem into chunks to have clearer and more specific objectives. This can be done repeatedly to further break down the problems and make my goals easier to meet and understand.

I have broken down the problem using an organisational chart which goes top-down.



(need to include some 1 player element to this if necessary)

I have broken down the problem into 4 main sub-problems that need to be tackled so the game can have the proper functionality. They are as follows:

- Controls

The controls of the game are necessary to allow the players to have control over the fighter movements and attacks. The players should have separate key presses on the keyboard in order that they control one of the characters. They must also be calibrated in a manner by which the key presses stay fixed for one of the warriors and is not based on the side of the screen it's on. For example, to control Fighter 1's movement, the W,A,S,D keys may be used anywhere on the battlefield.

- Game State

These states will allow for the players to have control over the different stages of the game. They are the means through which the user can transition to the next game state or stage. The menu game state would have buttons such as "Start", "Instructions" or perhaps "Exit". There will be a transition between the menu game state to the playing game state which is the mode where the fighting will take place (the core part of the game). If the player wishes to stop playing during this game state, he can transition (through a key press) to a paused game state allowing for the options to "Resume" or "Quit". Once the playing game state is finished (one player or the other depletes the others' lives completely), the game should

transition to the Game Over game state which should display the message “Game Over” and transition back to the Menu game state.

- *Players*

As this is a 2-player game, the fighters on screen will have separate animations and looks which can determine the style each player fights with as well as affecting the overall look of the combat. This will give the game a more artistic and interactive experience as the users will become familiar with their warrior and what works best for them.

- *Score*

Without the score, the game has no end point. The determining factors for the score the health bars of either warrior. Once the health bar depletes a live is taken from the respective warrior. This allows for the player to understand whether he's winning or losing the game and gives a sense of achievement when he takes all his opponents lives. Moreover, this means the player can be more dynamic in his tactics, going from attacking if he's in the lead, or slightly more defensive if he's close to losing a life. Once the lives are depleted, the game is over and the player with lives remaining wins.

Thinking logically

Thinking logically will allow me to approach the problems outlined in a systematic way using logic and problem solving.

- To deplete the other players health bar, you must land a successful attack. A logical algorithm must be implemented to ensure an attack is registered. To ensure an attack is registered correctly as being damaging, I will use selection, where the program will determine whether the attack was within range of the other players vector coordinates. This is thinking logically, as a decision must be made to determine the scoring, whether a successful attack has been made or it has missed.
- Another aspect of thinking logically, is that the controls are designed in a way that makes it intuitive for players when moving and attacking with their warrior. For example, when moving toward the right, the character should be facing towards the right and instantly switch when moving left. In addition, attacks should be thrown relative to the last “facing” position of the character. This further demonstrates logical thinking, as the game must work in a way that is logical and easy to play.

Thinking Concurrently

Thinking concurrently refers to managing multiple processes that are executing at the same time. This is quite fundamental to my game as it is 2 player – so one warrior moving, which is an independent task, shouldn't hinder the other warrior from moving as well. Else this would make the game extremely unfair, as whoever presses the key first is able to completely dominate the game. The algorithm within my code must be able to process inputs from each player concurrently.

Summary

Overall, the examples given have shown the features of my problem which can be solved by computational methods and why. These aspects which I have broken down and structured indicate that my problem is best solved through a computer program.

Software and Hardware requirements

In this section I will be discussing the software and hardware requirements for development and execution of the game. It is imperative to note that I have used the PyCharm IDE as a development environment and use the Windows 11 operating system.

Specific Requirements	Justification
Software	
Operating System: Windows, macOS, Linux.	These are the operating systems that Python files are compatible with. This is needed in order to execute the game as it is a python program.
PyCharm IDE	It is compatible with Windows, macOS and Linux and provides a development environment to code the game (includes tools like code completion and debugging).
Python	Python is the high-level language which Pygame is written with and hence is needed. It is essential as the functions and logic is mainly controlled by Python.
Pygame library	Pygame is the cross-platform library that is designed for coding games. It provides features that allow for graphics handling, input, animation, and sound which is very important in the development process.
Packaging Assets	These assets ensure that the player has the required files to run and play the game. So, images, sounds and other resources.
Executable file creators: PyInstaller	These allow for executable files to be created and run on multiple different platforms and devices without requiring the user to download Python!
Hardware	
Mouse	Required to select game modes and options within the game.
Keyboard	Required to control the character during gameplay.
Monitor/Laptop	This is the interface that allows the user to view the game.

Audio Device (Headphones/Speakers)	So, the user can hear the sound effects and background music included in the game.
1GHz Processor	In order that the executable code can be run.
Minimum 2GB RAM	Needed to run the application.
Few GB of storage	Needed for python and libraries/packages the scripts require to run.

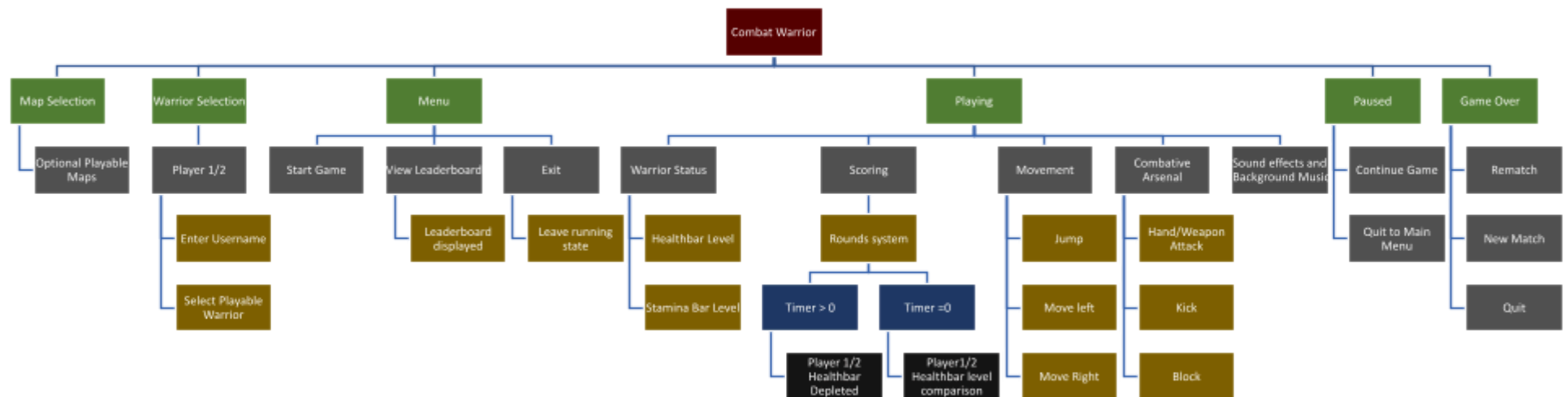
Success Criteria

	Criterion	Justification
1	Menu screen displayed with clear buttons including Play, settings and quit.	This is the first game state the player will encounter. The user must be prepared to play before entering into a "war" between the fighters. This means being able to adjust settings before playing.
2	On menu screen: have a view leaderboard option	This allows for the players to view who has the longest streak of wins as well as to view their own high scores.
3	Display a Player and map selection screen	To allow for the users to select their characters before entering into a fight. Also, the users can choose a background of their fight.
4	Display a box where each user enters a one-off username	So that the leaderboard can register a streak under an identifiable holder.
5	Give option for each player to select from at least 4 playable characters and 3 different maps	To provide variation in the game and allow for each player to experience a unique set of attributes and styles.
6	Display each players health and stamina bar (stamina bar with critical level).	The player must know how much health and stamina he has left as these are the metrics which are essential to win the round or defeat the opponent.
7	Display round indicators	To show how many each player has won in the respective battle. This is crucial as to win a "war", the player must win 2 rounds.
8	Display streak counter for each player	So that the player can know how many fights he has won consecutively and hence compete on leaderboard.
9	Have a 30 second timer	To make the rounds restricted by time and the player can know where they stand in the round with regards to time
10	Display each fighter in their starting position on either side of the screen.	To start the fight of with some distance and indicate the beginning of the round
11	Allow each fighter to move right, left and jump with separate keys on the keyboard	This movement should be gradual to challenge the players and add realism. Moreover, it is important as

		the player uses these movements to dodge attacks, create space and combine with attack keys.
12	Allow each fighter to punch, kick, and block and for these to be combined with movement for more specific attacks	So that each player can control the way their fighter attacks the other player and can use combinations to score a high kick for example.
13	Register and show a depletion in stamina for every attack and recovery	To balance the game and make it more tactically orientated. This means players can't spam a barrage of attacks with no rest as this is unrealistic. If stamina bar drops below critical level, the player must recover beyond this before attacking.
14	Register and show a depletion in health for damage taken via attacks	As this allows for players to know how much damage they have dealt or received and hence change game plans.
15	Round won if either health bar is depleted, or if time is up and one has higher health than the other	So that the indicators can be checked if this occurs, and the user can win the rounds and hence the battle. Player whose health bar is depleted loses the round. Player who has less health after time loses.
16	"War" ends once either player has won 2 rounds	This is to signify that the fight is over. The warrior who won 2 rounds won the fight.
17	Message asking for a rematch, new match, or exit	To allow for the player to build their streak and experience different fighting styles. Adds an element of competitiveness and variation.
18	Sound effects for blocking, punching, and kicking and taking damage	Sound effects will add depth to the game and make it more appealing to other senses. It can also give an indication as to what the opponent may be doing through sound and sight.
19	Background music for menu screen and fight	Adding background music that varies based on map selection gives each map and fight a unique feel. In the tiebreaker round, more tense music will be played to increase anticipation and tension.

Design

Systematic Decomposition of the problem



Why did I use the above process to systematically decompose the problem?

Due to using a top-down modular design, I have been able to easily organise my identified problems into smaller and smaller chunks. This will aid me before beginning the development process as I can look at each problem individually and understand their specifics. Moreover, using this approach allows me to use computational skills such as “Thinking ahead” to determine the inputs and outputs as well as “thinking logically” to see how parts of the program effect the program flow and logic. Overall, this has provided me a general structure and overview of my solution, allowing me to keep in mind the bigger picture goals of my game.

Explanation of the structure of Combat Warrior**Menu**

The menu screen will consist of the following components.

Start Game

This is the button (which will be part of the GUI) that will allow the player to begin the game. This transitions game states from menu to Map and Warrior Selection.

View Leaderboard

This allows the players the option to be taken to view the leaderboard.

Leaderboard displayed

The leaderboard displays the values in the database linked to the game in graphically within the game.

Exit

This is an option given to the player which causes the game to go from a running state to not running hence closing the GUI and game itself.

Leave running state

This calls the script which causes the game to exit the running state.

Warrior Selection

The warrior selection occurs before the map selection allowing for the player 1 and 2 to choose from the available playable characters.

Player 1 or 2

Players 1 and 2 can select from the characters.

Enter username

There will be a box appearing on the warrior selection screen prompting each player to enter a username – player 1, then player 2. This will be used as a reference in the leaderboard.

Select Playable warrior

There will be an interface with small images of characters. The name and description of each character will be displayed as the player toggles through the player select.

Map Selection

This game state allows for the selection of a map for the fight to occur.

Optional Playable maps

On the GUI, there will be an interface for scrolling through and selecting the maps. There will be a button to confirm the selection of their map.

Playing

This game state is the most essential. It features the actual fighting in the game and the scoring system used to determine the winner.

Warrior Status

There are two main components regarding a warrior's status within the round.

Health bar Level

The health bar on the screen will illustrate how much health is left or has been depleted by the contrast of a different colour when health is lost. So, a full health bar will be green, and when health is lost, some of the green will be replaced by a red to indicate that much lost health. The less health a warrior has, the less attacks it can take before depleting.

Stamina Bar Level

The stamina level of a fighter is indicated on the displayed stamina bar. This level will vary increasingly and decreasingly depending on the status of the fighter. The stamina level of a fighter will decrease upon it attacking and recover upon not doing so (playing in a passive or defensive manner).

Scoring

The scoring system in the game is dependent how many rounds a fighter has won. If the player has won two rounds – which the game checks for at the end of each round, a message is displayed declaring the winner of the game.

Rounds system

The rounds are timed. Whether the fight is won or lost is dependent on which warrior won 2 rounds – as it is first to win two rounds. However, the round being won is dependent on the timer and health bar levels which has been explained below.

Timer > 0

While the timer is greater than 0, the game checks if the value of the fighter's health has reached 0.

Player 1 or 2 Health bar depleted

If the opponent's health bar is depleted, the user with remaining health will win the round. The game displays whether player 1 or player 2 won the round unless one of the players has reached 2 rounds won.

Timer = 0

If the timer has reached 0, the game compares the value of either fighter's health to declare a winner of the round.

Player 1 or 2 Health bar level comparison

The player with a greater amount of health wins the round. A message displays the winner of the round unless either player has reached 2 rounds won.

Movement

There are 3 fundamental ways which a character can move. The fighter's movement is controlled by the respective player.

Jump

When a key is pressed – dependent on which character being moved – the fighter will jump. How does it work? The fighter will have a set position on the ground and when it is commanded to jump (in this case exclusively), it will move in the y-direction negatively (this is because when programming the screen, to move down, this is a positive increase in the y coordinate) with a set velocity and once it has reached its maximum vertical displacement, returns to its original position vertically. There will be an animation that occurs during this motion.

It is also worth noting, that when the fighter jumps while moving in a horizontal direction, a parabolic motion will entail. This can possibly be modelled with a quadratic.

Move left

When a key is pressed – again, dependent on which character is being moved – the fighter will move in the specified direction with a certain set velocity. An animation will occur upon this command.

Move right

Look at move left section. The same applies. It is important to note that I mean the movement relative to user. In actuality, the fighter would be facing each other.

Combative Arsenal

This consists of attacks that either fighter can execute. This is controlled by the player.

Weapon Attacks

These are the primary source of attacks. Each fantasy character will have its own unique weaponry with animations. There will be various types of weapon attacks for each fighter.

Kick

This is a less significant secondary attack. They can be used as a variation and to catch opponents off guard.

Block

This is a defensive move which essentially stops the full damage of an attack.

Sound effect and Background Music

In the game, sound effect will be implemented upon successful attacks and movement. Moreover, there will be sound effects when interacting with the GUI to make the user experience more responsive. Adding sound effects will increase depth to the game.

Background music will be added for all game states. Depending on the map selected, the background music for the playing game state will be distinct. It will also increase in intensity to increase tension in the deciding round of the fight.

Paused

In the playing game state during rounds, the user will be able to pause the game until resumed. This will transition from playing to the paused game state.

Continue Game

This will be a button as an available option on the paused menu. It is a function which causes the game to transition back to the playing game state.

Quit to Main Menu

This option allows users to exit and save the game. Streaks are saved and displayed in leaderboard if applicable. There is then a transition back to the menu game state.

Game Over

This is a game state which occurs after the fight is over. It gives players the option to rematch, have a new match or quit the game. The streak of either player is saved at any of these points.

Rematch

This option saves the state of the initially selected warrior and map, and transitions into an initialised playing state. This gives either player a chance to fight again and change their strategy, etc.

New Match

This option will cause a transition into the warrior selection game state and the usual transitions will take place before fighting. However, the game streak will appear on either players side of the screen to indicate how many consecutive games have been won.

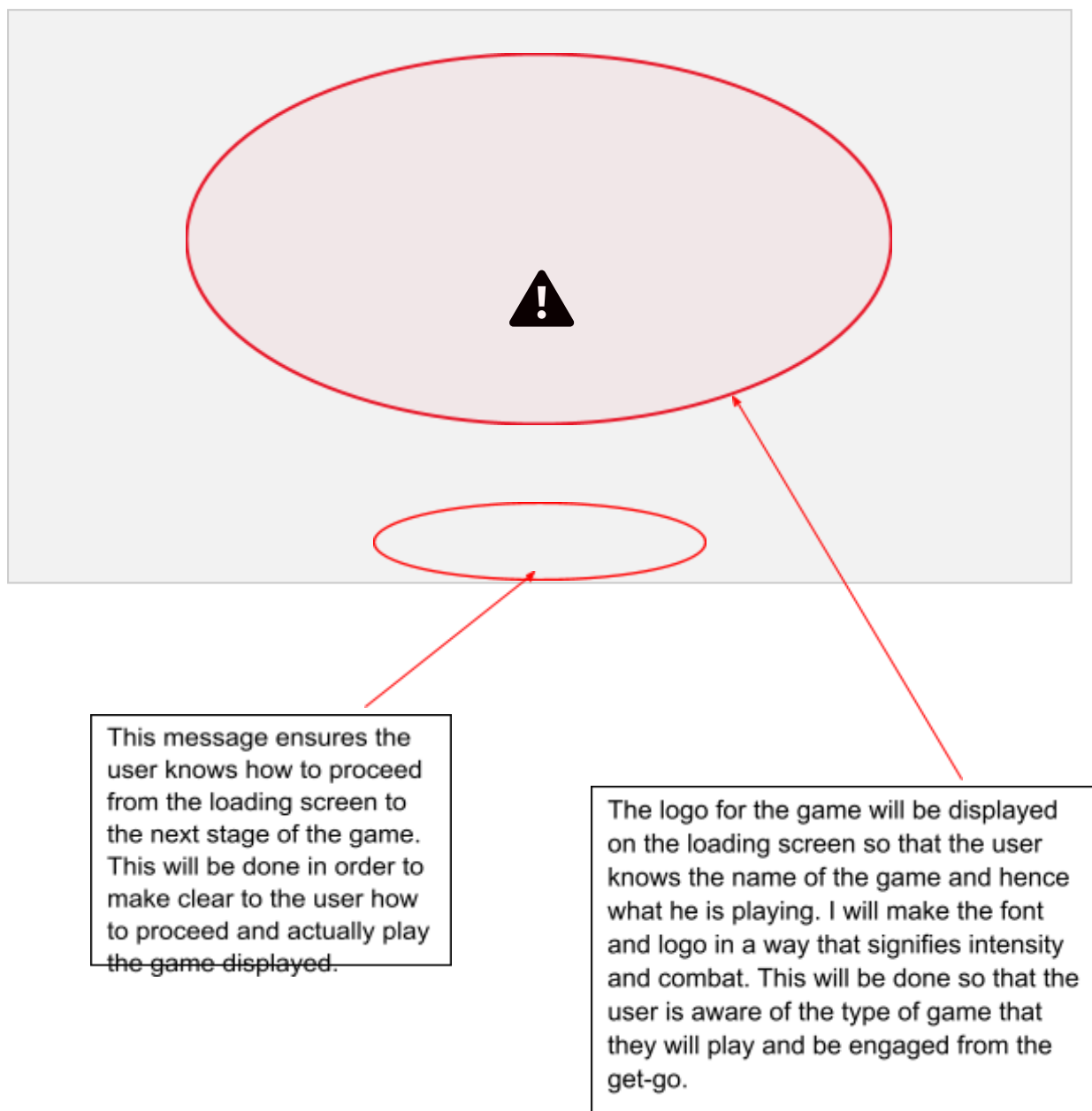
Quit

This option when selected will quit to the main menu game state. The players streaks are saved and if high enough, are saved to the leaderboard.

Usability features and concept art of Combat Warrior

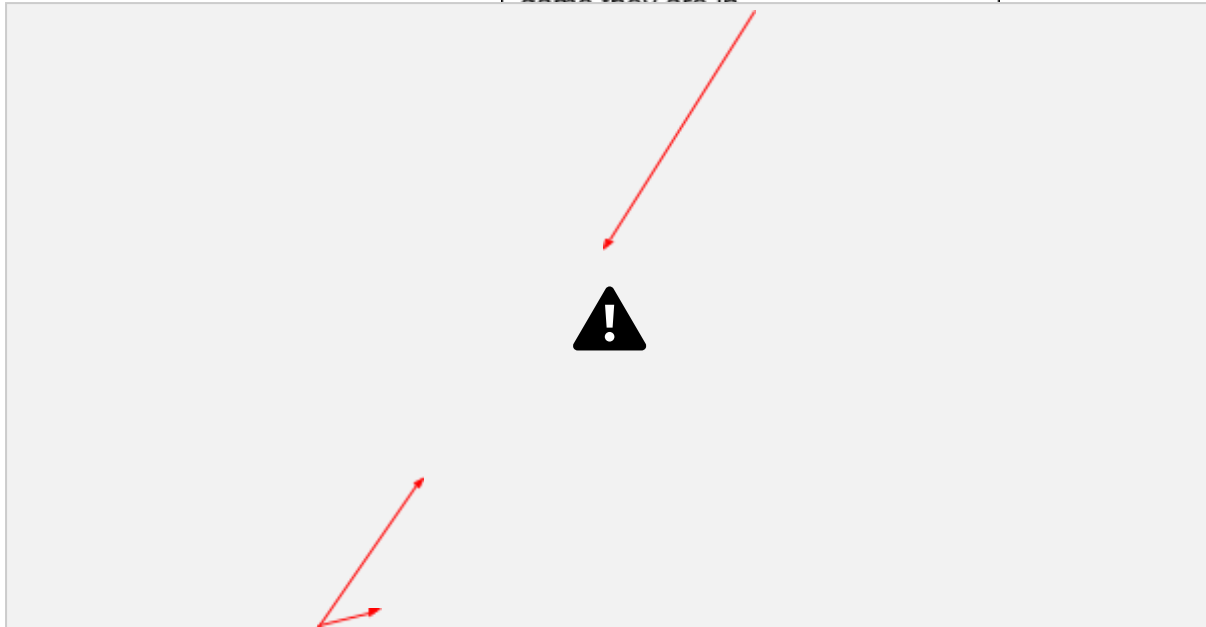
In this section I have included some concept art of the screen designs for my game. I have explained the key usability features of each screen and intend to justify the logic behind them. The screens are laid out logically and chronologically, as they would appear in the game, so as to provide a better understanding of the user experience.

Loading Screen



Main Menu

There will be a big heading to indicate to the user that they are on the menu page of the game. This will be done to make it easy for the users to know what stage of the game they are in.

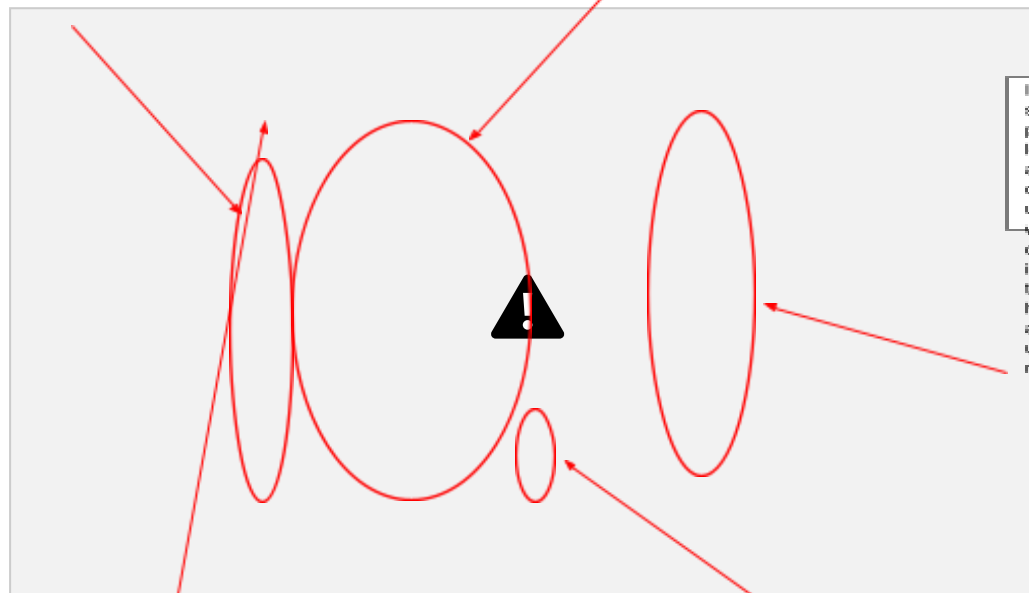


On the menu screen, three buttons are displayed so that the user can select either one based on their preference. This will be done to serve as a visual cue that guides the user through the options in the menu. Moreover, it provides a clear visual representation of each option allowing the user to quickly identify the available choices.

Leaderboard displayed

In my design, I have included numbers next to the users on the leaderboard. This is displayed so that the user viewing the leaderboard is aware of the current rankings. It allows users to see how their performance may compared to others and where they stand.

The usernames of the respective players will be displayed next to their number, so it is clear to the viewer who holds the position and score. It fosters a sense of competition and motivation to improve, which keeps players more engaged with the game.

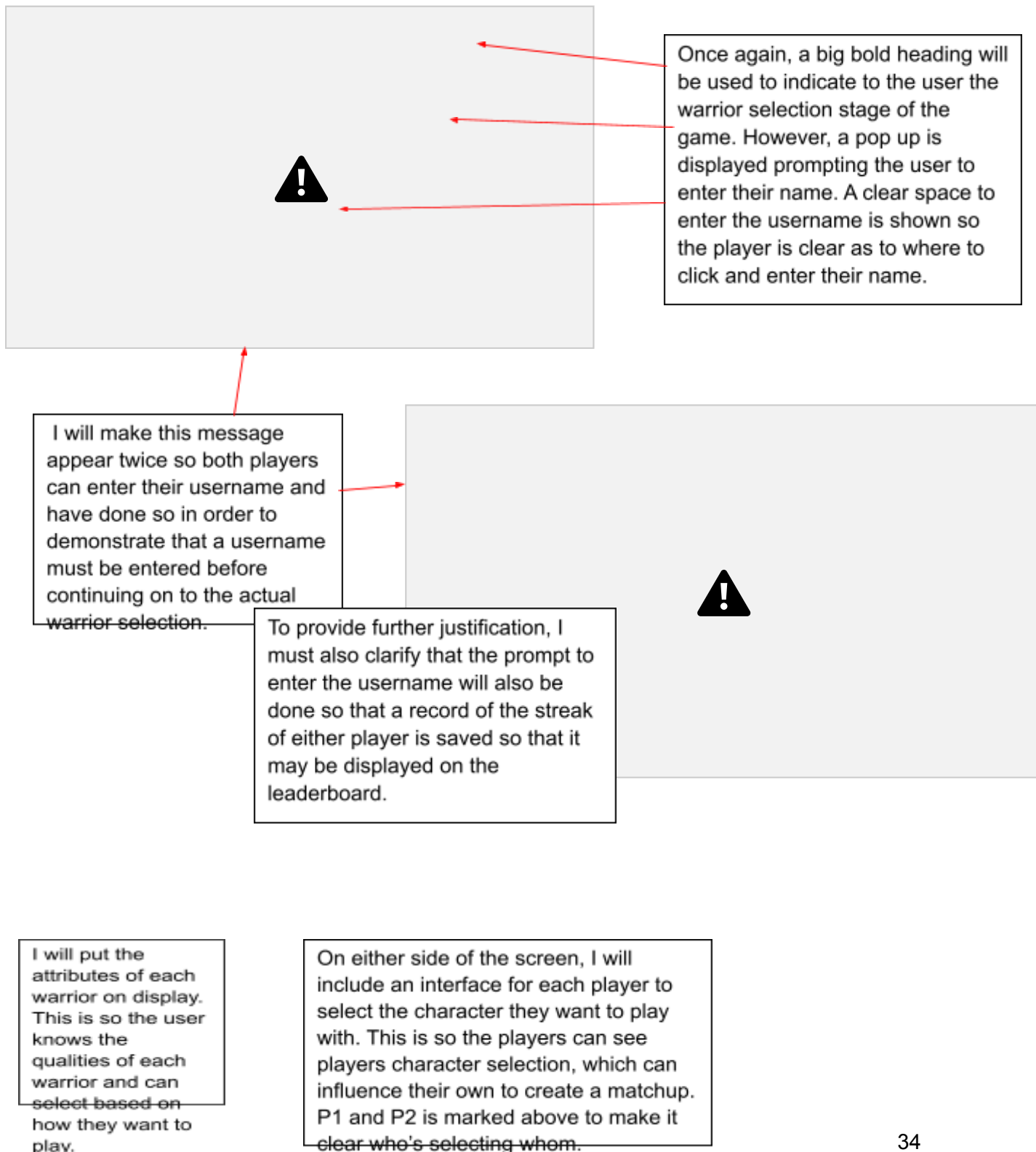


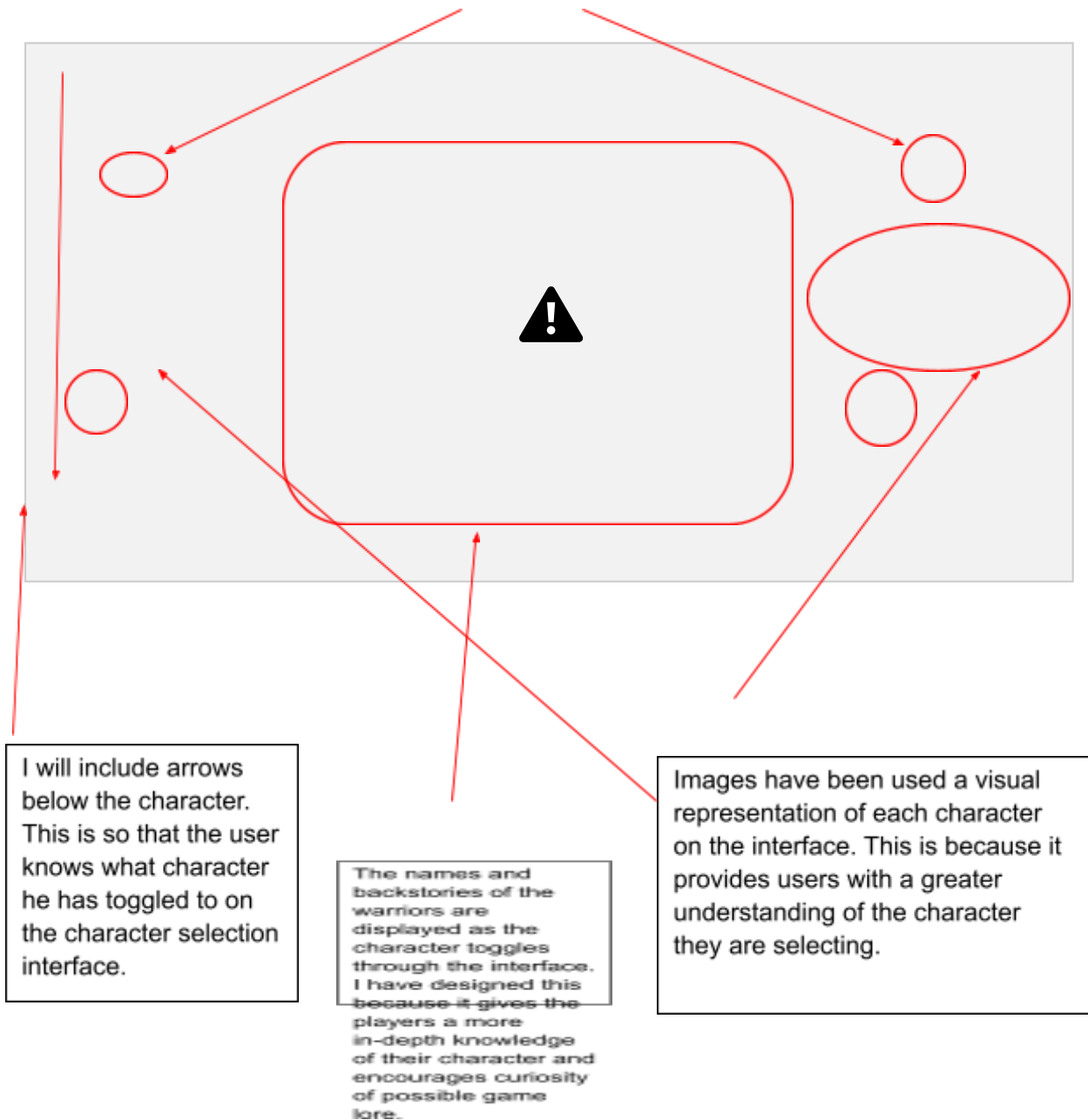
I will include the streaks which the players on the leaderboard achieved in a column next to the username and warrior. This will be done in order to indicate the scores the high-ranking players achieve, and users use this as a milestone.

There is a title on the leaderboard in order to indicate from a different state of the game. This will be done so that the user understands they have transitioned game state and are viewing the leaderboard.

Next to the username, the warrior with which the player obtained the streak with is displayed. This is because it allows the player to view what other users of the game are gaining streaks with, perhaps inspiring them to select a particular character. Moreover, if one warrior seems to be very dominant, it allows users to be competitive and use different characters.

Warrior selection





Map Selection



I will display an interface for the users to select their map. As this is a two-player game, the users can decide by toggling to the different displayed maps in the interface. This will be done so that the user knows what the map looks like and can collectively decide.

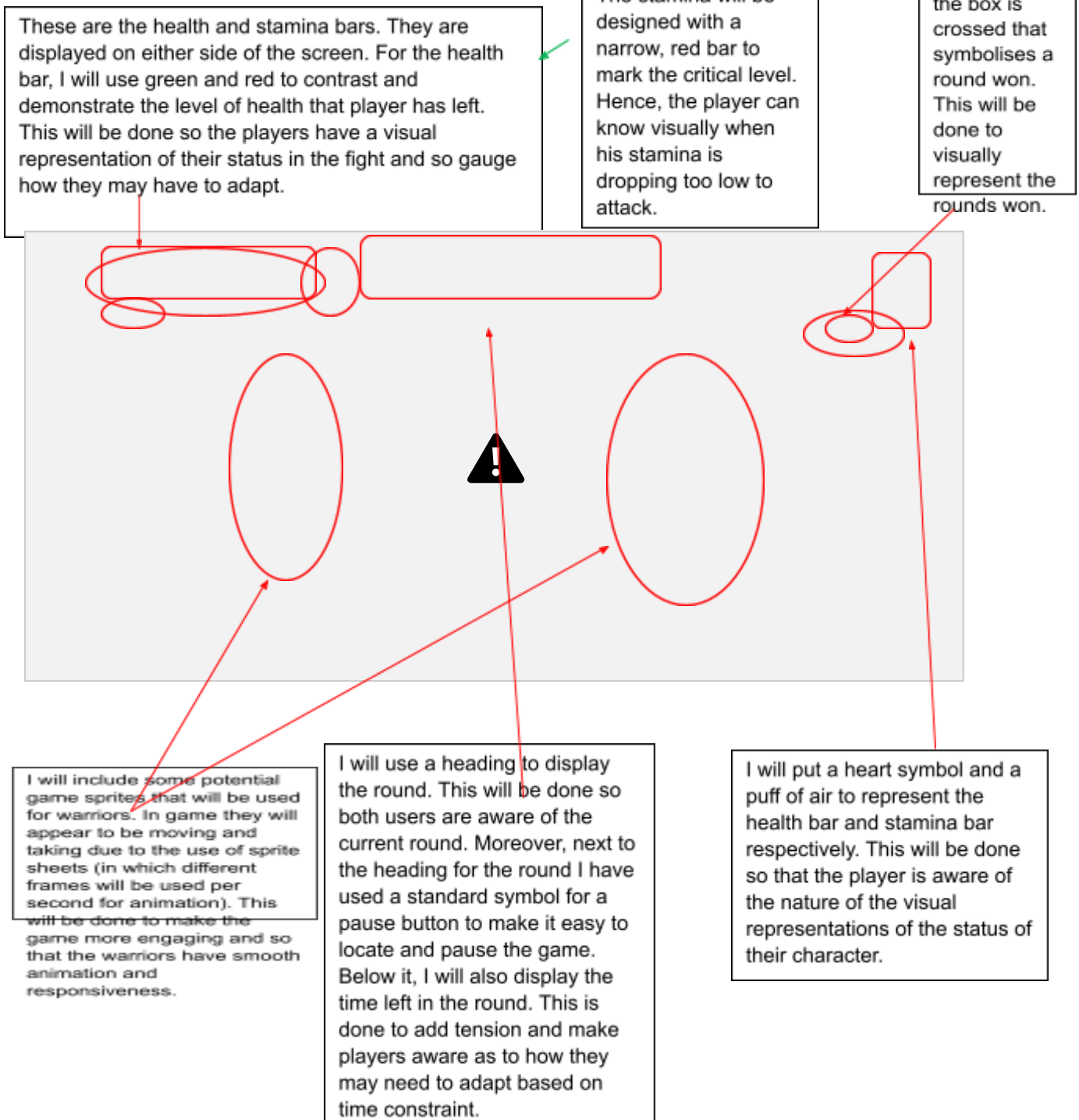
I will use an arrow to indicate which map the users have toggled to. This is so the users have a visual cue as to what map they are selecting and makes it more interactive as well as clear when selecting the map.

Playing

In this section I have displayed a few different states of the playing game state. I will explain the streaks slightly later.

Situation – Mid Round

This screen design shows mid-round of the second round of the battle. There are 35 seconds left on the clock.



Situation – Player 2 wins

round





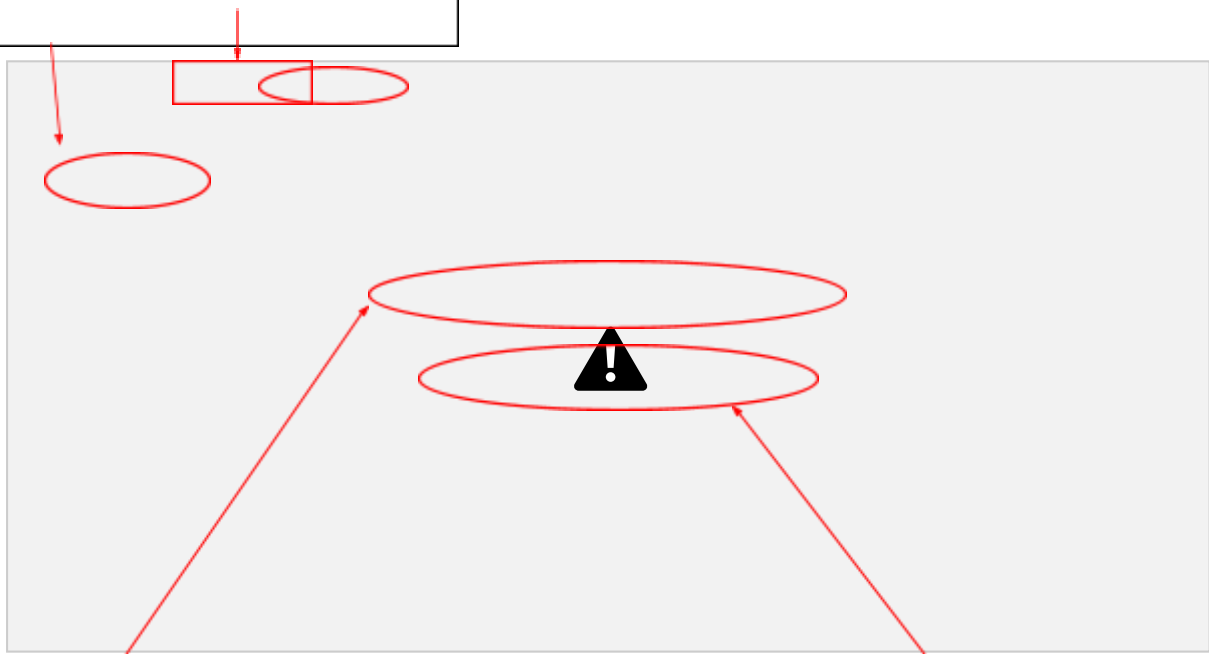
In this scenario, player 1's health bar has been completely depleted. This is shown through a completely red background for the health bar. This shall be done to visually represent to the user the poor status of their warrior.

It will show the player's health bar as red, indicating that the player's health is low. This will be done to visually represent to the user the poor status of their warrior. The health bar will be red when the player's health is low. This will be done to visually represent to the user the poor status of their warrior. The health bar will be red when the player's health is low. This will be done to visually represent to the user the poor status of their warrior.

When one player has won the round, either player 1 or player 2, I will display a message that says which player won the round. I have done so to provide immediate feedback to users allowing them to see the outcome of their efforts in the game. Moreover, it is done to clear any confusion as to who won the round in a close call.

Situation – One Player wins the war

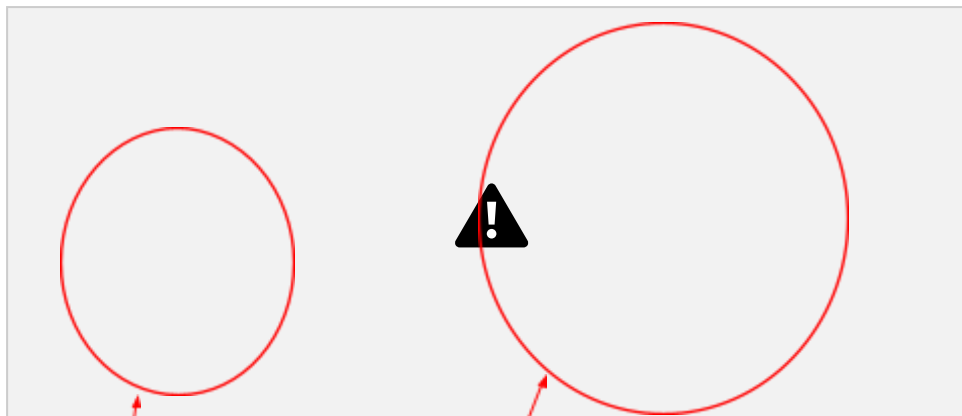
The streak will be displayed above the players health and stamina bar. As can be seen, the streak has been updated and will be displayed briefly before the game over state is displayed. I will do this, so the players have a fleeting sense of achievement and anticipation for another match to improve their streak. The same has been done for the round indicator.



At the end of a round, if the timer reaches 0, a message informing the users that time is up is displayed. This will be done to clear any confusion as to when the round is over, and to make both users aware that a decision is made based on a comparison between health bars.

A message will be displayed informing who has won the war as one player has won 2 rounds. I will add this so that the players are certain who has won the war. It also provides a sense of achievement to the winner through a visual cue.

Movement and Player controls

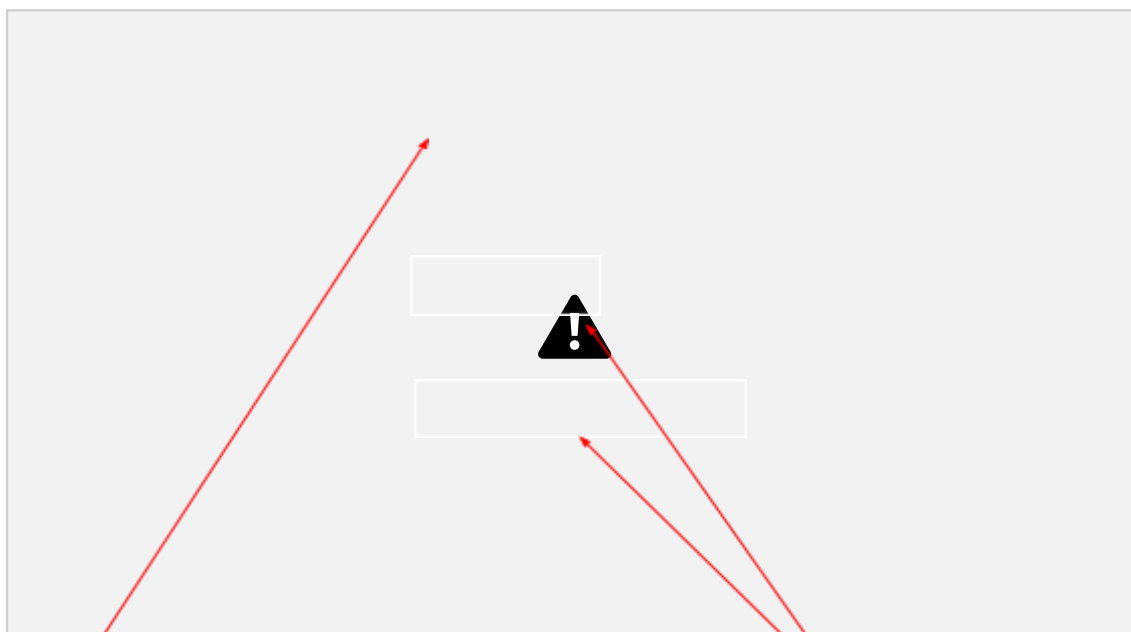


On this left side of the keyboard, which I highlighted green, will be the controls for player 1 (which will initially be on the users left). I will do so as this is most intuitive for the users as the left controls should control the player on the left. The same has been done for the right side, the player controlling player 2, as this is most intuitive.



The left click button will be used to select from the on-screen buttons

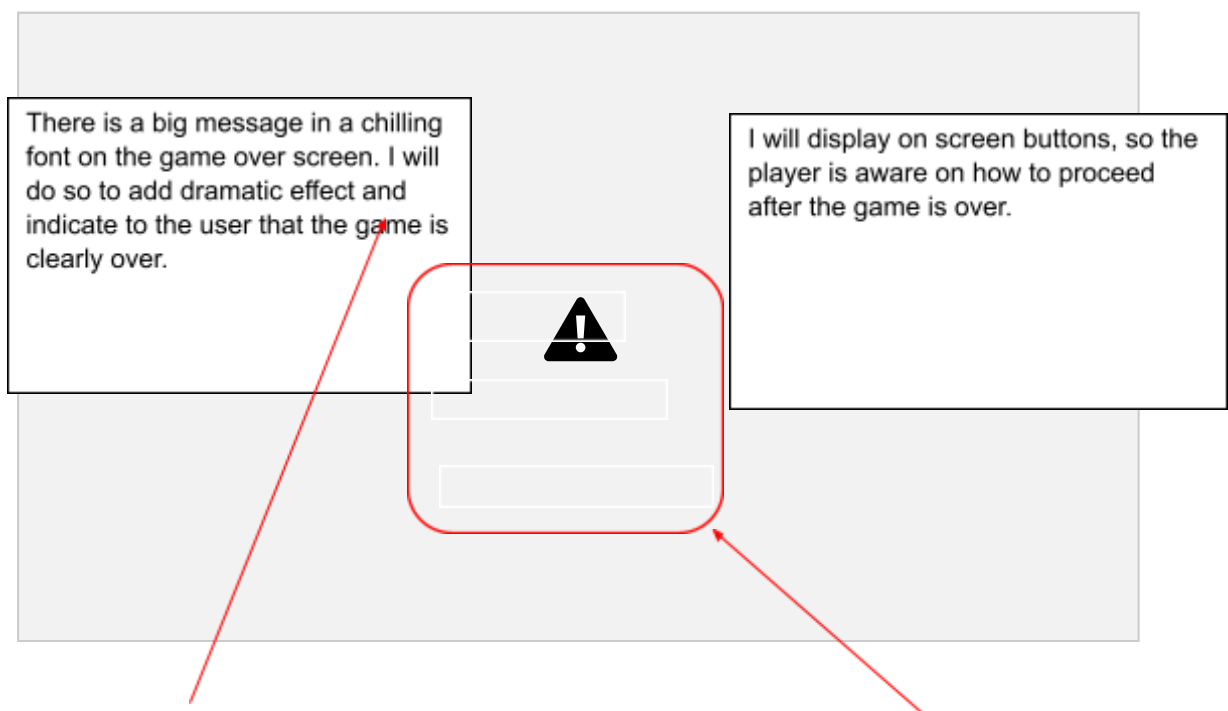
Paused



When a player clicks the paused button during a round, they will transition to this screen – a paused menu. I will display a paused heading which is bold and clear to inform the players that they are now in the paused game state.

I will also include boxes around the options to resume and quit and save. This makes it clear that it is an on-screen button.

Game Over Screen



Key Algorithms

Game state transitions for Main Menu

Pseudocode	Justification
main_menu is True define function startGame(): main_menu is False	Main menu is Boolean because the game can only be in one state at a time. The main menu game state needs to be false before I can call a function to transition to the next game state.
if main_menu is False then warrior_selection()	I used a conditional statement as I only want the function to be called if a condition of main menu being false is met.
define function viewLeaderboard(): main_menu is False if main_menu is False then display_leaderboard()	The functions for viewing the leaderboard have a similar logic. The function should only display the leaderboard if the main menu is false first because otherwise the states would collide and overlap.
define function exit_game(): terminate program	If exiting the game, the program should terminate or break because the game should no longer be running. Pygame has in built functionality for this.

Username input

Pseudocode	Justification
define function username_input(user): enter_username_prompt(user)	This algorithm prompts the users to enter the usernames until a username is entered.
store_username_inDB(user)	This function takes the usernames entered when prompted and stores them in the database.
warrior_selection is True	After the functions have been processed, the warrior selection game state variable can be set to true which allows for it to be called in the game loop.

Warrior Selection

Pseudocode	Justification
define function warrior_selection(user): warrior_selection is True display_warriors() user_selects_warrior(user)	Allows users to choose from a set number of warriors. The parameter user is passed in so that the function user_selects_warrior accounts for the individual user.
if both users selected warrior, then warrior_selection is False	The if statement sets a condition which transitions to map selection game state.

map_selection is True	
-----------------------	--

Map Selection

Pseudocode	Justification
<pre>define function map_selection(): display_maps() selects_map()</pre>	Displays the maps and allows the users to select the map.

Playing state algorithm

Pseudocode	Justification
<pre>if playing is True then handle_movement (warrior) handle_fighting_arsenal (warrior) update_warrior_status() update_round_indicators() check_round_winner () update_timer()</pre>	If the game is in the playing game state, then the warrior status, round winner and timer all need to be updated. These functions are called only in this state.
<pre>display_winner() game_over()</pre>	To display the winner and transition to the game over state.

Handling movement

Pseudocode	Justification
<pre>define function handle_movement (warrior): if jump_key_pressed: warrior.jump() if left_key_pressed: warrior.move_left() if right_key_pressed: warrior.move_right()</pre>	The parameter warrior allows for the argument to be passed in when calling the handle movement function. This means a specificity for the warrior. Function allows players to jump, move right and left on command.

Handling fighting arsenal

Pseudocode	Justification
<pre>define function handle_fighting_arsenal(warrior): if attack_key_pressed: warrior.attack() if kick_key_pressed: warrior.kick() if block_key_pressed: warrior.block()</pre>	Function allows for specified warrior to attack, kick and block upon specific key presses.

--	--

Pause game

Pseudocode	Justification
define function pause_game(): playing is False if playing is False then display_paused_menu() handle_pause_menu_input()	This algorithm sets the Boolean variable playing to false in order to display the paused menu. The paused menu should only be displayed if playing is false.
else playing is True	The paused menu inputs such as button presses should be handled. Setting the playing variable to true will act as a resume. This save having implement a resume game logic.

Game Over

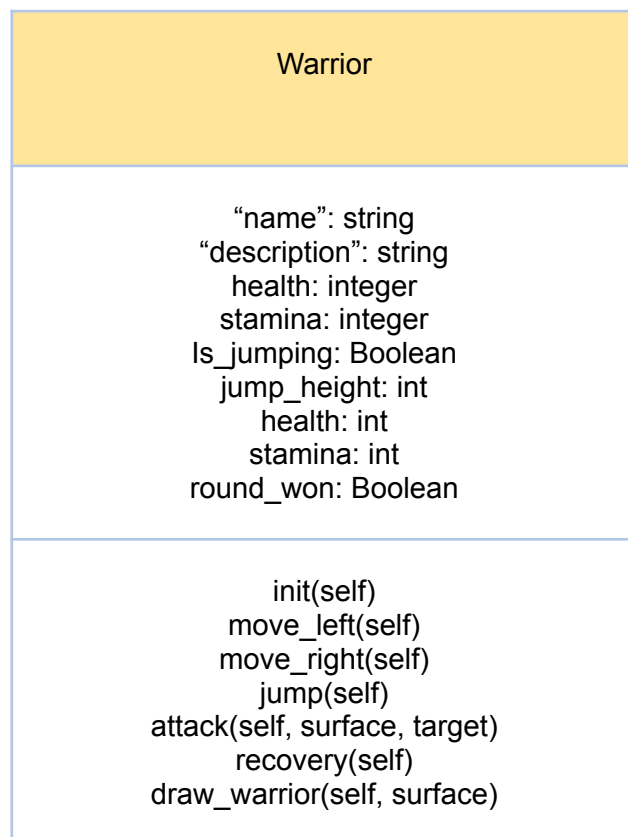
Pseudocode	Justification
define function game_over(): display_game_over_menu() handle_game_over_input()	This displays the game over menu and handles the relevant inputs if the game over function is called.

Handling game states in game loop

Pseudocode	Justification
while game_is_running is True: if main_menu is True then display_main_menu()	As main menu is set to true by default, the main menu will be displayed first before anything else. Hence this function automatically called at the start of the game.
else if paused is True then pause_game() else if game_over is True then game_over() else [insert playing logic]	The relevant states features and menus should be displayed upon the game states being true.

Key Variables, data structures and Classes

Warrior Class Diagram



Variable Tables

Warrior Object variables:

Variable/Attribute	Data type	Purpose	Justification
name	String	Represents name of warrior	To identify each warrior and give them their own unique name.
health	Integer	Stores the current health level of the warrior	Determines the players status and standing within the round

stamina	Integer	Stores the current stamina level of the warrior	Determines warriors' stamina and influences warrior's attacks
jump_height	Integer	Dictates the height or y coordinates which the warrior displaces when jumping	To give the warrior a certain fixed jump height
Is_jumping	Boolean	Stores whether the player is jumping or not	To determine whether the player is jumping or not
Is_attacking	Boolean	Stores whether warrior is attacking or not	To determine whether the player is attacking or not
x, y	Integer	Allows us to dictate the x and y position of each warrior	So, each warrior has its distinct starting position and movement can be tracked

Game state management:

Variable/Attribute	Data type	Purpose	Justification
playing	Boolean	Indicates if game is being played	Controls whether game is being played or not in game loop
paused	Boolean	Indicates if game is paused or not	Determines whether to set playing to false and display pause menu
view_leaderboard	Boolean	Indicates whether the game leaderboard is being viewed or not	Controls whether leaderboard is being displayed
round_timer	Integer	Represents remaining time in round	To manage the duration of a round and cap it.
round_won	Warrior	Stores winner of current round	Determines which warrior wins the round
current_map	Map	Stores currently selected map	Tracks map which will be used in the war

Other Global/Local variables in main file:

Variable/Attribute	Data type	Purpose	Justification
SCRN_WIDTH SCRN_HEIGHT	Integer	To set the dimensions of the game window	So that the game window is visible

clock	Pygame.time.Clock	To control frame rate of game and manage time related things	Can be used to fix a framerate so the players don't move according to refresh rate of device.
start_countdown	Integer	To set the time before entering the playing state	Allows players to be prepared before playing.
attack_sound jump_sound	pygame.mixer.Sound	Represents playable sound object that can be loaded from a wav or suitable file	Gives users an audio indication that an attack or move has been made
bg_music	pygame.mixer.Sound	Represents playable sound object that can be loaded from a wav or suitable file	Gives a feel to the game and element of engagement

Iterative Test plan

In this section I will look at the relevant test data to be used during the development process.

I have broken the problem down into several milestones and for each have defined what is being tested and the expected outcome.

Milestones

1. Warrior Movement and Attacks
2. Playing – Fighting and Round system
3. Paused interface
4. Menu interface
5. Warrior and map selection interface
6. Sound effects and background music
7. Game Over interface
8. Leaderboard display

Milestone 1: Warrior Movement and Attacks

Test Number	What is being tested and relevant inputs?	Expected Output
1	Game window	Suitable Game window appears with background image.
2	Instances of warriors	Two distinguishable warriors appear on screen.
3	Warrior horizontal movement with keys a, d and RIGHT, LEFT	Both warriors move left and right on key press.
4	Warrior stays within bounds of the screen (horizontal)	Warrior does not move off the screen when it is moved to the edge.
5	Warrior vertical movement with w and UP key.	Warrior jumps. Displaces from origin in minus y direction and returns to 0 relative to floor level.
6	Warrior stays within bounds of screen vertical and does not go below the floor	When jumping, warrior does not drop below floor level.
7	Warrior attack. Input keys controlling each player	Warrior performs an attack on key press.
9	Sprite sheet for animation of attack, movements (MAY BE IMPLEMENTED AFTER FIGHTING AND ROUNDS)	Extracted frames repeatedly looping cause illusion of smooth movement

Milestone 2: Playing- Fighting and Round system

Test Number	What is being tested and relevant inputs	Expected Output
1	When player is in range, attack registered	Health variables depletes to indicate decrease in health
2	Health bars at top left and right corner of screen	Should appear for both players to visually represent decreases in health
3	Health bars at start of round	Display only green health level to indicate 100% health.
4	Player 1 Health bar depletion	Depletes upon receiving successful attack from Player 2. Green bar reduces to reveal red.
5	Player 2 Health bar depletion	Depletes upon receiving successful attack from Player 1. Green bar reduces to reveal red.
6	When player attacks. Stamina affected.	Value in stamina variable decreases when the player attacks. Increases upon rest.
7	Stamina bars top right and left corner below health	Stamina bars appear below health bar to visually represent increase and decrease of value of stamina.
8	Stamina bars depletion	Drains for respective player as they attack. If drops below critical level, player can't attack.
9	Stamina bars recovery	Recovers stamina for respective player as they aren't attacking.
10	Round indicators displayed in top left and top right corner below health and stamina bars	Should fill with a colour based on who won the round. Winner of round gets box filled.
11	Timer count	Should decrease as fight goes on second by second
12	Timer reaches 0 (not final round)	Round should end and compare health levels of both warriors. Message to display this visually.
13	Health bar completely depleted	Round should end with player with remaining health being declared the winner.
14	Stamina bar reaches below critical level	Player with critical stamina should not be able to attack until it recovers stamina.
15	Streak counter displayed above player status bars	Updates once the match is over based on who won. Stored in separate file or database.
16	Player wins 2 rounds	Fight is over. Display streaks briefly and visually displays winner of match. Transition to game over state.

Milestone 3: Paused

Test Number	What is being tested and relevant inputs	Expected Output
-------------	--	-----------------

1	User input causing the playing game state to be paused	Pause any loops currently being run for the playing state on key press
2	Paused menu	When paused, a menu is displayed with options to resume or quit.
3	When user resumes. (from paused menu)	The paused loops should now be resumed, and the game state should now be back to the playing state.
4	When user quits. (from paused menu)	The game state should transition from paused menu to the main menu. Playing data is saved.

Milestone 4: Menu Interface

Test Number	What is being tested and relevant inputs	Expected Output
1	The background for the menu screen appears. Game transition from loading screen.	The background for the menu screen fills up the entire interface.
2	Buttons displayed correctly and in order.	Buttons appear on the menu.
3	Mouse clicks on the GUI buttons.	Transition to the selected game state with relevant loading screens.

Milestone 5: Warrior and map selection interface

Test Number	What is being tested and relevant inputs	Expected Output
1	Box prompt for entry of user surname before warrior selection.	Box appears in centre of screen prompting user to enter username.
2	Keyboard presses to enter username in box prompt.	Key presses will display respective characters.
3	Box prompt disappears upon user clicking enter key (if enough characters)	Box prompt disappears and player 2 is presented with box prompt.
4	Box prompt does not disappear until valid username inputted including at least 4 characters and not already taken.	Box prompt stays until valid username input.
5	Database registers valid entered username.	(back end) username added to database.
6	Screen is split into two sides for player 1 and 2 on GUI.	Player 1 side on left, player 2 side on right, indicated by markers for player 1 and 2.
7	Character interface displayed and mouse hovers on warrior images.	Animation to show selection, character summary and attributes appear.

8	Mouse clicks on warrior images	Sound effect and cue as confirmation.
9	Both warriors selected and confirmed.	Transition to map selection.
10	Mouse clicks on map images.	Sound effect and animation. Transition to playing state.

Milestone 6: Sound effects and background music

Test Number	What is being tested and relevant inputs	Expected Output
1	Background music for main menu state	Background music should play when player is navigating menu and selection.
2	Background music in the rounds	When entering the playing state intense background music should be played.
3	Background music in the final round	Music should intensify or change
4	Upon attack by either player	Sound effect upon attack depending on the warrior
5	On screen button press	Sound effect upon presses of these buttons

Milestone 7: Game Over interface

Test Number	What is being tested and relevant inputs	Expected Output
1	Fight has ended	Game state should transition to game over state
2	Upon game over transition	Game over menu with 3 on screen buttons new match, rematch or save and quit.

Milestone 8: Leaderboard display

Test Number	What is being tested and relevant inputs	Expected Output
1	Upon fight ended	Streak count should be stored in separate file or database
2	Leaderboard interface	Should display visually, values stored in separate files or database.
3	Leaderboard layout and ranking	Player with highest streak should be at the top. Username displayed next to streak.

Post-development and robustness testing of Game

Robustness test	Method of testing	Justification
Local Player Playtesting	Organize local playtesting sessions with pair of players. Observe engagement, competitiveness and enjoyment level while playing.	To evaluate the games engagement and fun factors is a real gaming environment.
Animation smoothness and visual appeal	Record gameplay and share it with friends and family. Obtain feedback on how visually appealing the animations are and whether they are positive or negative.	To see whether one of the games aim of being a smooth, cleaner version of the original game has been achieved or can be improved.
New player experience testing	Gather at least one new player in age demographic to see how quickly they learn the game mechanics and obtain feedback on difficulty level of the game.	To see whether the games controls and walkthrough is easy to understand or challenging.
Boundary testing	Test whether warriors in game do what they are meant to and cannot be manipulated or broken. Test what happens if multiple keys are pressed at the same time and if random keys are pressed all at once.	In order to test for any holes in the coded solution and also check for any loopholes which may affect the way the game is played.

My Development Diary (Coding the Proposed Solution)

In this section of my documentation, I have included milestones and the code I wrote in order to achieve these. I have documented testing, screenshots, and user feedback for a full break down of the code.

As there are many parts to developing each milestone, I have explained each section of the milestone for clarity.

Milestones:

1. Warrior Movement and Attacks
2. Playing – Fighting and Round system
3. Paused interface
4. Menu interface
5. Warrior and map selection interface
6. Sound effects and background music
7. Game Over interface
8. Leaderboard display

(28/02/2024)

Milestone 1: Warrior Movement and Attacks

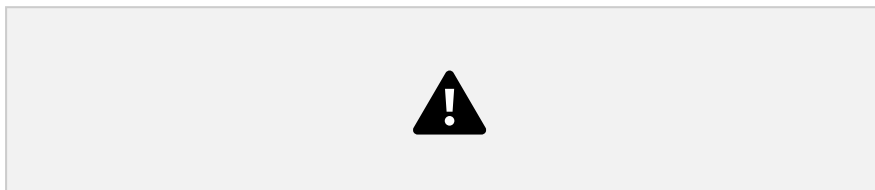
This milestone is the most fundamental to the game as it introduces the fighter's mechanics and movement. I have also included development of the game window as this must be done before creating instances of the warriors.

Creating the game window and setting up the main game loop

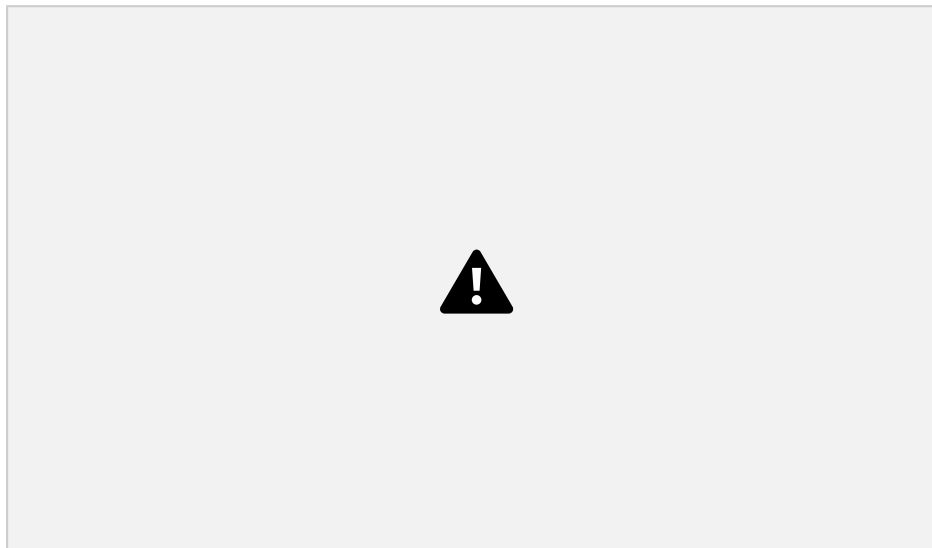
For this step, I aim to create a skeleton of the game which will be easy to build upon as I go along. I will create the basic game loop and also display the game window and the background.

Iterative development and testing

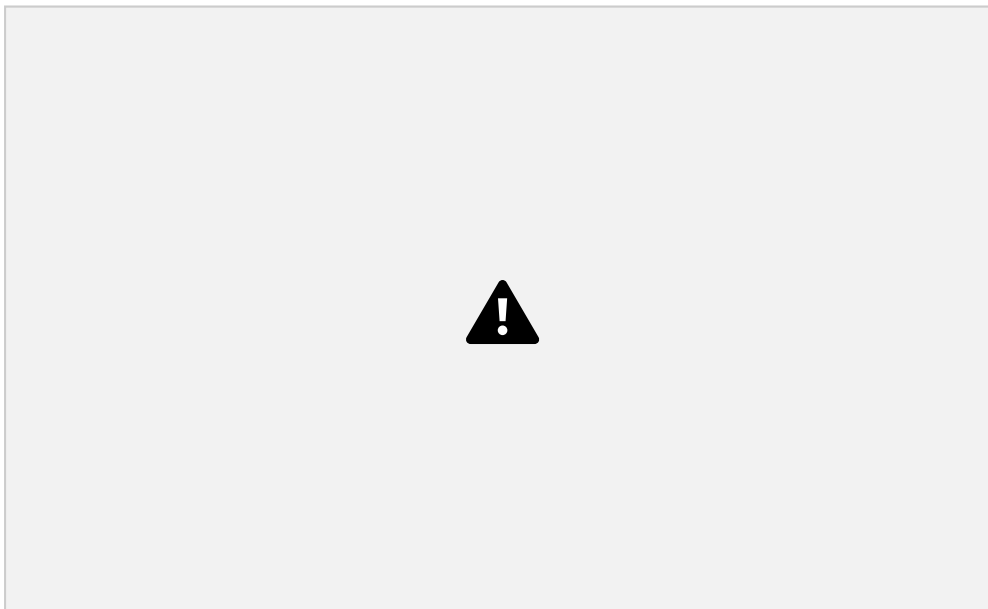
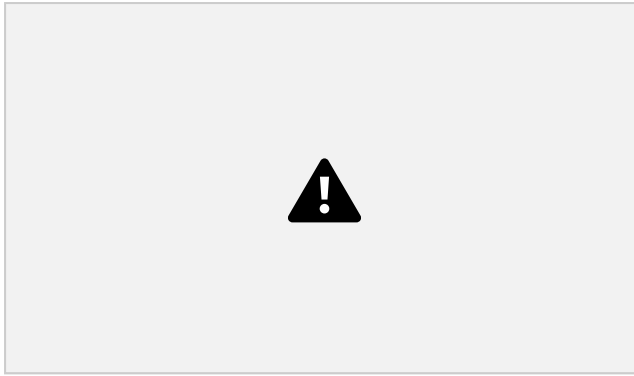
Initially, when I manage to get the game loop working and screen dimensions defined, I tested that the game window was being displayed correctly. When I did, I decided the screen width was too narrow and would not give the players enough area to move back and forth. Hence, I changed the Screen width from 800 to 1000. This makes the screen wide enough, which is good for users to see the screen clearly.



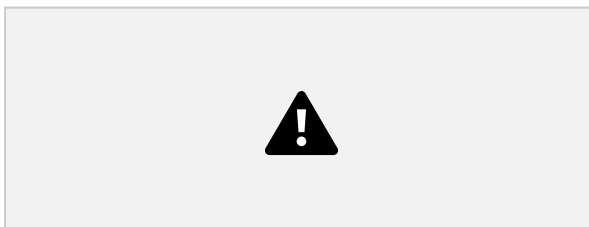
Below is screenshot of blank game window (1000 x 600) with the Combat warrior caption.



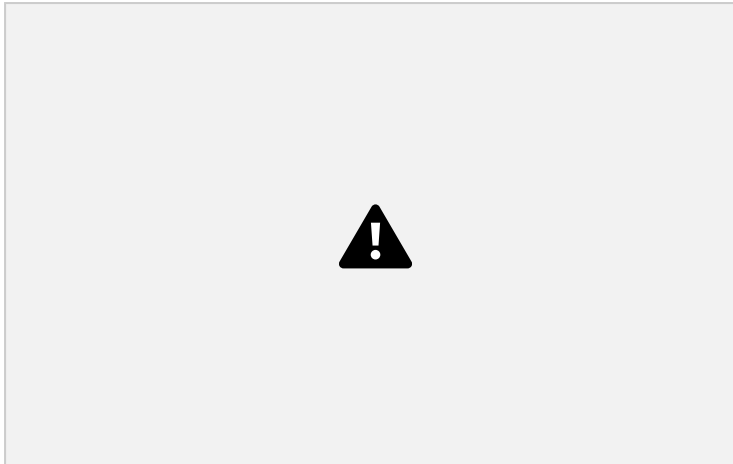
When I tried to put a background image, nothing appeared.



This was due to me not updating the Pygame display window. Hence, I used `pygame.display.flip`.

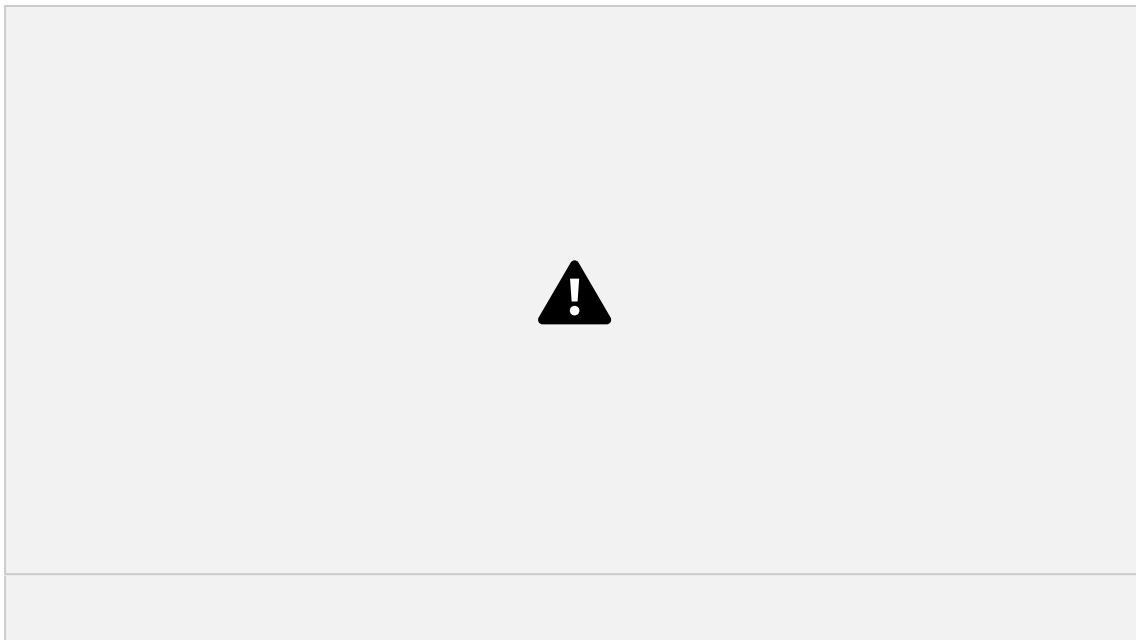


This resolved the issue:



Test Number	What is being tested and relevant inputs?	Expected Output	Actual Outcome
1	Game window	Suitable Game window appears with background image.	A suitable game window appears with background image.

Code and explanation



Before I defined the screen dimensions or the main game loop, I imported the Pygame modules of the Pygame library through the import Pygame statement. I also imported the

sys modules because this allows me to access command line arguments which was useful particularly for coding the game.

I then initialized the Pygame modules in order to get everything started and have access to the Pygame functionality.

Next, I defined the screen dimensions and the caption for the game window. I saved the width and height as variables "SCRN WIDTH, SCRN HEIGHT" which could be changed later.

In order to keep the game running, the main game loop is used. Essentially, the game window will only close if the running variable is set to false. This is triggered by clicking the cross at the top right of the window. Now, a blank window appeared (black background) which did not close until commanded to.

I loaded in a background image using the pygame.load function in order to load the image from my folder into Pygame.

I defined a function called draw_bg which scaled my background image as it was initially too big for my screen – hence I set it to my screen width and height – as well as using the blit function, which took two arguments – the image and the x and y coordinates to blit it, allowing me to actually display the background image. This function was called within my main game loop.

User feedback

My stakeholder, Mateen thought that the background image for the game was very cool and had a mysterious vibe. He did agree that initially, the width of the screen was too narrow and hence it was a good move to make it wider.

Creating Instances of warriors

For this iteration, I aim to create and display two instances of warriors on either side of the screen to represent player 1 and player 2.

Iterative development and testing

I created my class with a constructor and a method to draw my warrior. I realised that there was an issue.

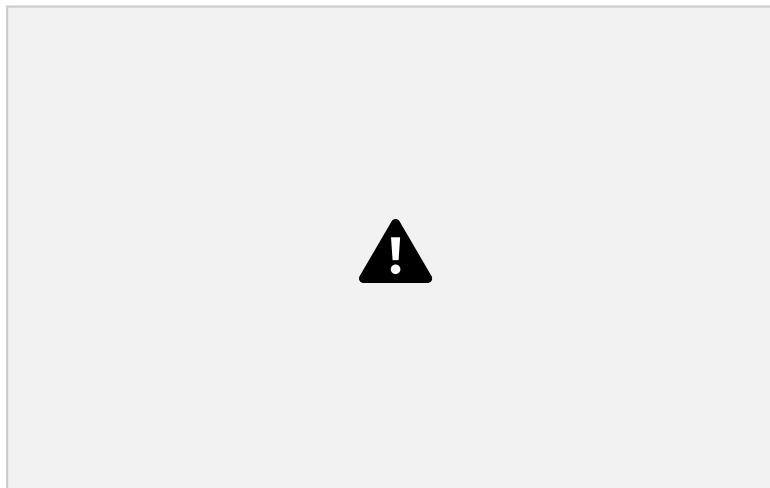
I tried to fix this by including a colour in my constructor method for the rectangle, but this didn't change anything.



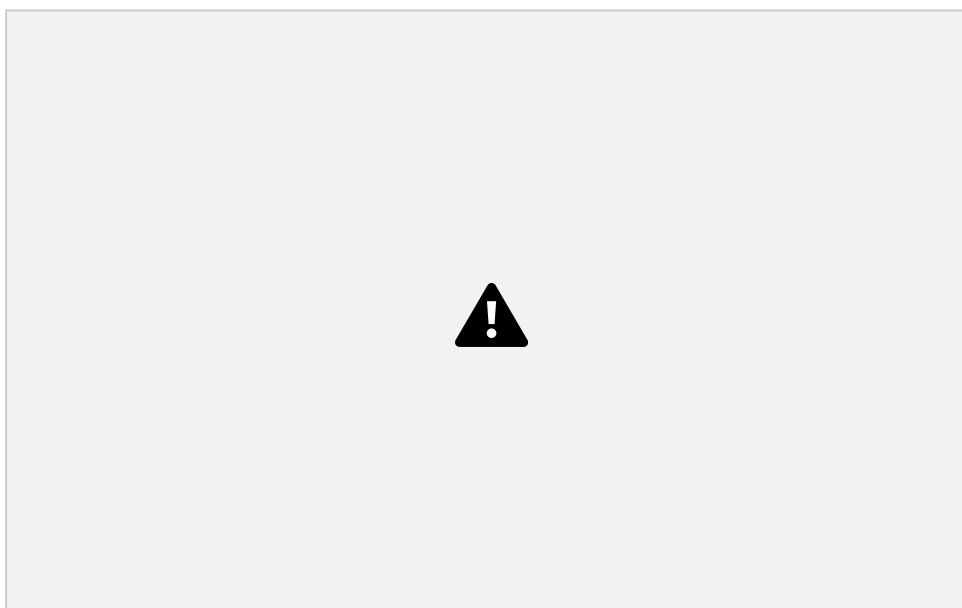
Then I realised that a rectangle with a color was not a Pygame surface, so I was using the wrong function. I then changed this by making my draw method draw a rectangle with arguments surface, color, and rectangle.



This fixed my issue and two instances of my fighters appeared on the screen.



I changed the position of the warrior 2 to an x coordinate of 700.

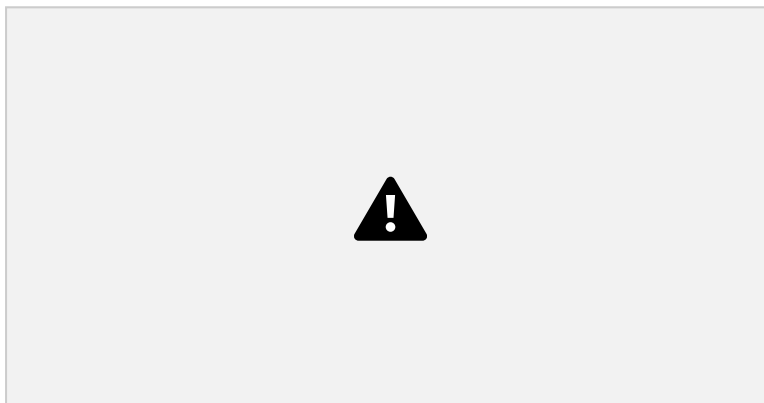


Test Number	What is being tested and relevant inputs?	Expected Output	Actual Output
2	Instances of warriors	Two distinguishable warriors appear on screen.	Two different color rectangles (instances of warriors) appear on screen.

Code and explanation

I decided to create a separate python file which handles the warriors' attributes and methods which I can call in my main file. This allows for my code to be more maintainable and also allows me to make changes to the class and hence multiple instances in my main file.

warrior.py file



In my warrior file, I defined the constructor method or init method which had attributes x, y, width height and color. The reason was because I decided to make the warriors rectangles initially, rather than loading sprite sheets and working on animations. I wanted to tackle the fundamental mechanics and gameplay before adding

these.

I also defined a method "draw_warrior" which drew the warriors to the surface which is defined in the main file.

Before I could create instances of my warriors, I imported my Warrior class from the warrior python file in order to access its functionality.

combatwarrior.py file

Then I created two variables which stored instances of what I called warrior 1 and warrior 2. Their methods were then called within the main game loop.



User Feedback

Mateen didn't have much to test at this point in the game, however he helped to adjust the positions of the warriors to an appropriate starting distance.

(29/02/2024)

Warrior Movement and boundaries

I aim to make the instances of the warriors which I made and add movement for both of them. They should move with the set key presses respectively.

Iterative development and testing

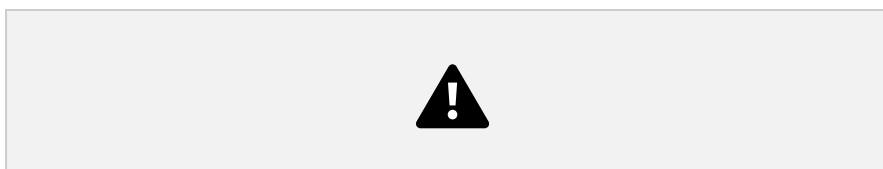
I successfully made the warriors move right and left on key presses. Defined attribute velocity in my constructor method.

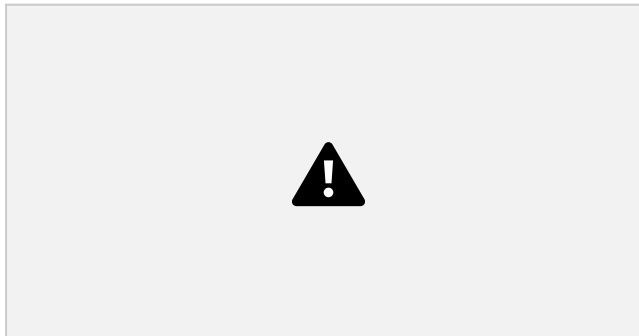
However, they moved far too fast.

There were two options: either to reduce the velocity, or to cap the frame rate of the game. If I reduced the velocity, depending on the device, this may be too slow or too fast. If I cap frame rate, the movement will stay fixed.

So, I created a clock object and made a global variable called FPS.

Then I made the clock tick at the fps.

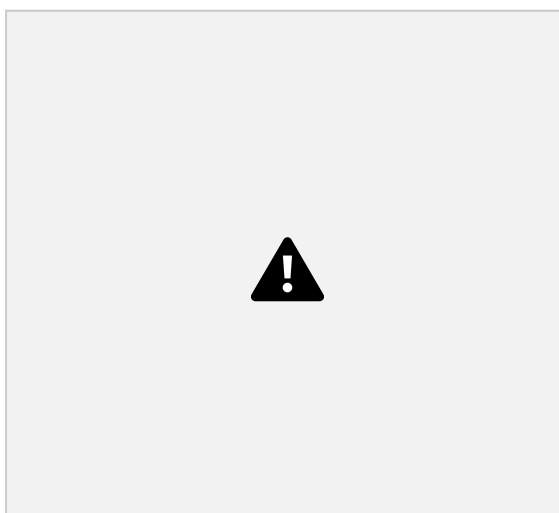




I wrote the function to make warrior jump. But on key press nothing happened:



I realised that there was nothing in my code to set the is jumping variable to true. So, it didn't work as expected. Hence, in my main file, I said if warrior jumps, call the function which sets is jumping to true. Then we actually jump.



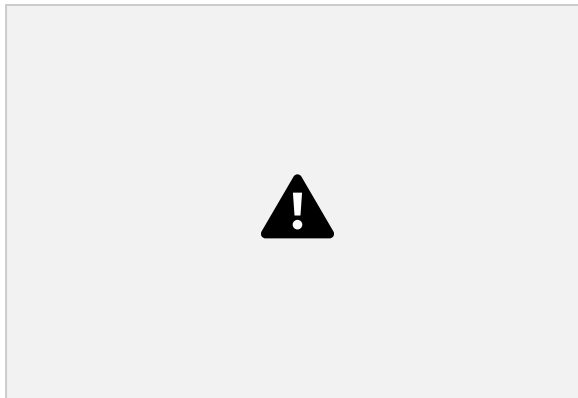
This has resolved my issue. They jump.

But they jump too high and rapid. So, I multiplied my jump height by 0.75 before squaring it.

Test Number	What is being tested and relevant inputs?	Expected Output	Actual Output
3	Warrior horizontal movement with keys a, d and RIGHT, LEFT	Both warriors move left and right on key press.	Both instances of the warrior move horizontally on key press.
4	Warrior stays within bounds of the screen (horizontal)	Warrior does not move of the screen when it is moved to the edge.	The warrior stays within bounds of the screen.
5	Warrior vertical movement with w and UP key.	Warrior jumps. Displaces from origin in minus y direction and returns to 0 relative to floor level.	The warriors jump correctly and return to floor level set.
6	Warrior stays within bounds of screen vertical and does not go below the floor	When jumping, warrior does not drop below floor level.	The warrior does not drop below floor level in any case.

Code and explanation

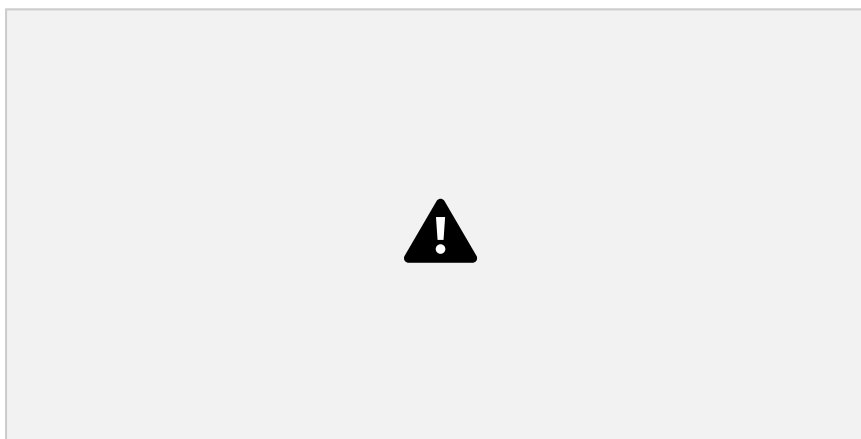
[warriors.py]



In my warrior class, I added some attributes for the instances of my warrior. The velocity defines the speed, or the x coordinates the warrior moves across.

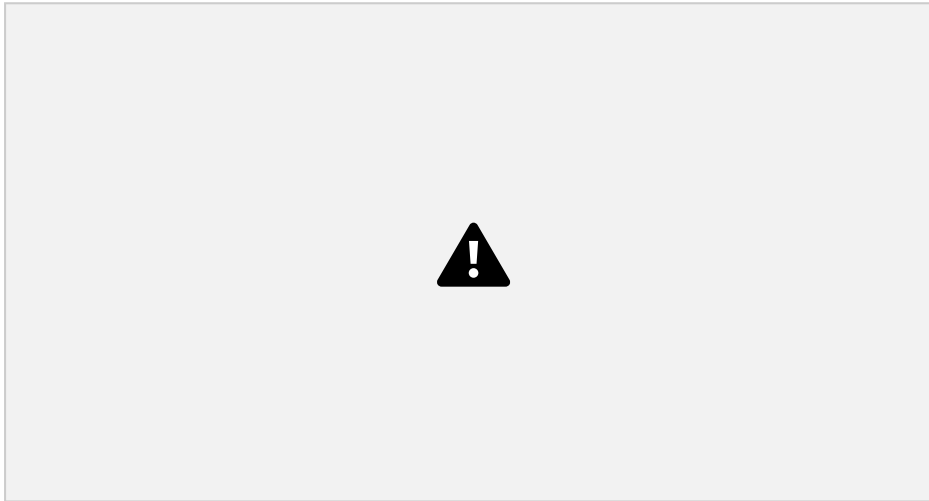
To create methods for jumping, I defined two attributes – is jumping and jump height.

Below in the method jump, the jump will only start if is jumping is set to true. This is done so that the jump only occurs on key press. The function make_jumping_possible is used to set the is jumping attribute to true. I called this in my main function on key press.



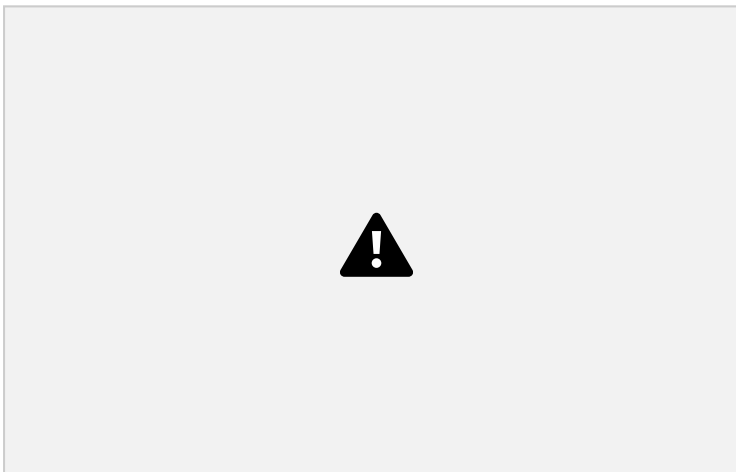
In order to move left and right, the warriors x position (represented by the rectangles x position) must decrease or increase by the velocity set in the constructor method.

To avoid moving of the sides of the screen, if the left side of the rectangle (`self.rect.left`), becomes less than 0 (hence moving of the left edge of the screen), the velocity changes to increasing. Hence, the velocity will no longer be subtracted from the x coordinate of the player. A similar logic is applied for the right edge of the screen.



The code nested within the if statement of line 42, controls the vertical movement of the warrior. First, it checks if the warriors jump height is above -10. If so, it adjusts the objects vertical position via a quadratic formula which is meant to simulate gravity, hence gradually decreasing the jump height. It also updates the jump height by decreasing it by 1 each time.

combatwarrior.py



I used a series of if statements within my main game loop in order to call the relevant warrior method upon key press.

For example, when the d key is pressed, `warrior_1.move_right()` is called so that the functionality is available.

User feedback

My user thought the jumping and horizontal movement was functional and at a good pace. However, Mateen suggested that the warriors moved across and got to the edge of screen to

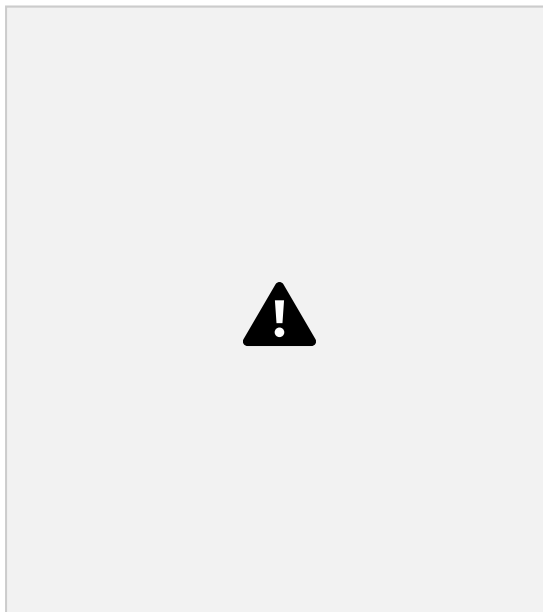
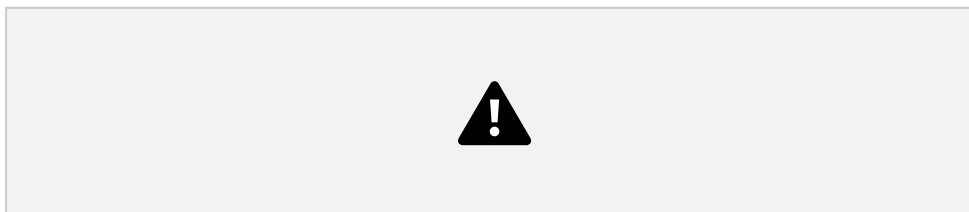
fast. He suggested the area of fighting was perhaps a little small or that the warriors were too big.

Warrior Attacks on key press

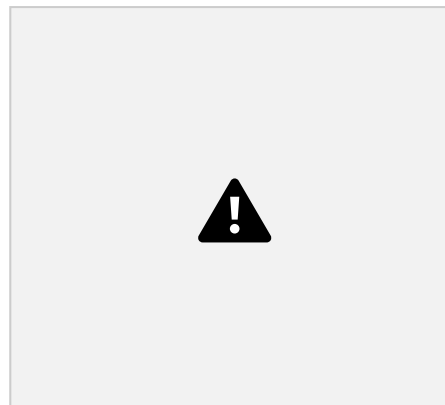
I am aiming to simply register an attack on key press. The collision and scoring logic will be implemented in the next milestone.

Iterative development and testing

I tried to implement the attacks in this way at first. Created attribute name and attack type. This way I can see who's attacking and with what attack.



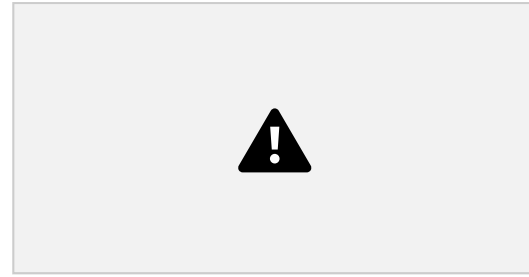
But I got an issue where when I pressed the key for either player, it registered the attacker and the attack used correctly, but it seemed to have registered multiple times per press.



(01/03/2024)

I then decided to remove these bits of code and think of another solution.

I tried to implement a check of the event type key down. This was to ensure that the command was being initiated upon key down. I managed to resolve my issue of multiple attacks being registered and got it to work successfully!



Test Number	What is being tested and relevant inputs?	Expected Output	Actual output
7	Warrior attack. Input keys controlling each player	Warrior performs an attack on key press.	Warrior attack registered once upon key press as needed.

Code and Explanation

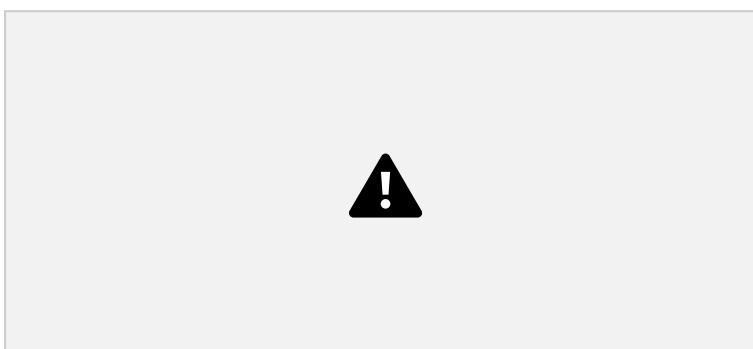
warriors.py



The “attack” method at the moment simply outputs which warrior has attacked, and with which attack type. F strings were used in order to pass in the name of the warrior in the game loop.

combatwarrior.py

In my controls, I added keys for each warrior which if pressed, the attack method will be called.



While the game is running, I am checking for another event type which is key down. This ensures that the attack only occurs on key down (ensuring for one attack at a time rather than multiple).

User Feedback

Mateen was happy with the development of the game so far and agreed that focusing on the fundamentals of the warrior was most important before implementing any attacks. He was now anticipating that attacks registered, and the rounds system be built.

(01/03/2024)

Milestone 2: Playing- Fighting and Round System

With this milestone I will implement the round system and fundamentals of how the rounds are managed and how the game is won.

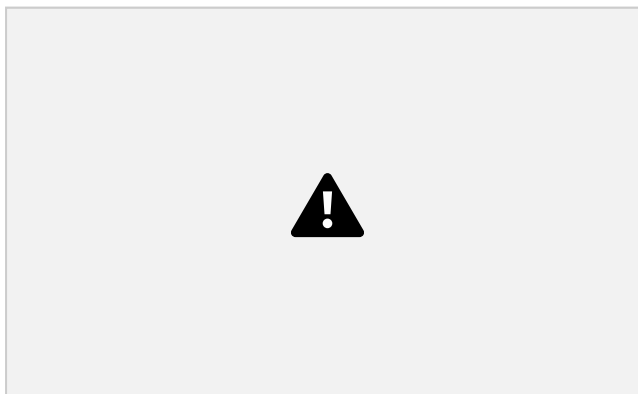
Attacking and collisions

For this step, I will now make it so when a player's attack collides with the target, the target will lose a set amount of health.

Iterative development and testing

NOTE: Before implementing this, I smoothed out the controls for the jumping of either warrior. This way the warriors don't continuously jump too fast. Again, this allows for one jump to occur at a time.

Code:



To register the attacks within a certain range I decided to draw another rectangle. If that collides with the warrior, that means an attack should be registered.

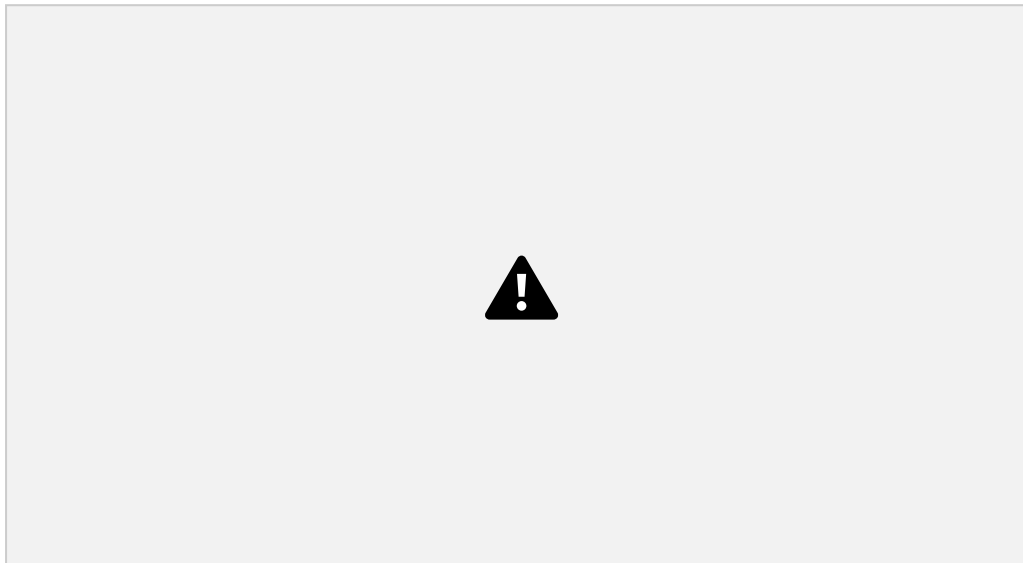


However, when running this, I received an error related to my main file.

I had forgotten to pass in the argument in my main file for attack.

I resolved this error and passed in the argument for the surface. But another problem arose. The attacking rectangle wasn't appearing upon key press and also the range wasn't a factor in the registration of the attack.

I realised that my background was drawing on top of my rectangles. So, I drew this before the main game loop.

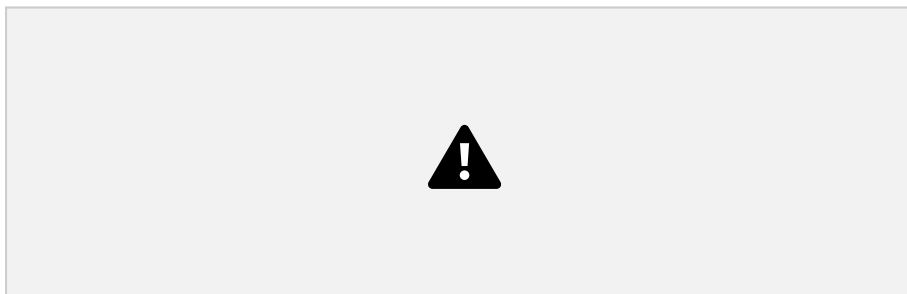


Now, both warriors had attacking rectangles.

However, there was an error which meant that the warrior could only attack to the right side.

I decided to implement flipping. This means that when the target rectangles centre is at an x coordinate less than the attacking warriors centre coordinate, we need to flip. This allows the players to attack both ways.

So, I defined a function called face correctly.



Then I used the self.flip in my attack method to draw the attacking rectangle correctly. Essentially it shifts the x coordinate of the attacking rectangle to the left side of its center should self.flip be true. Otherwise, that whole bracket is ignored.



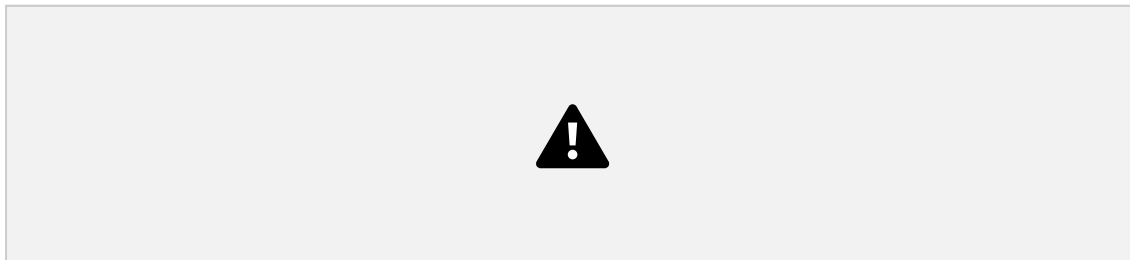
Test Number	What is being tested and relevant inputs	Expected Output	Actual Output
-------------	--	-----------------	---------------

1	When player is in range, attack registered	Health variables depletes to indicate decrease in health	When the player attacks, if his attack is within range of his target, it is registered.
---	--	--	---

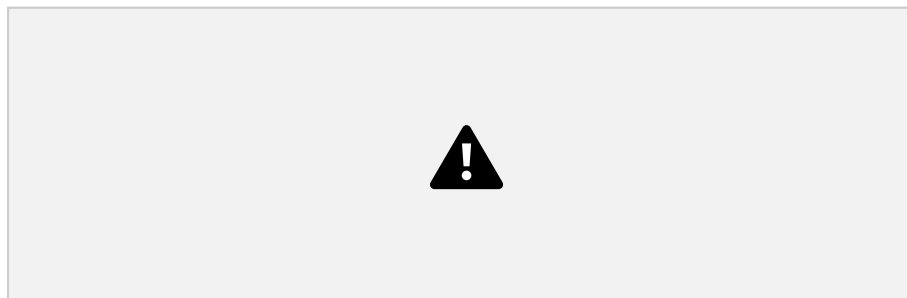
I will start working on the health when developing the health bars and hence have only caused it to register through a print statement.

Code and Explanation

warriors.py



In order to visually indicate an attack was occurring, I set the local variable “attacking_rect” to a rectangle which appeared from the centre of the warrior (self.rect.centerx). The attacking rectangle has the same dimensions except it is 1.5 times wider and a different color. The if statement attacking_rect.colliderect, essentially checks if the attacking rectangle is in range with a target. I changed the argument for colliderect to a target rectangle.



This method takes the parameter target. This code ensures that when the warrior passes their target rectangle (which

will be different for both warriors of course) then flip is set to true, and this causes the attacking rectangle to go towards the correct side or the side where their target is.



I changed my attacking rectangle to take these arguments instead. This essentially is the same, however, when self.flip is true, the attacking rectangle will be drawn to its left. Otherwise, it will be drawn from the centre as usual.

combatwarrior.py



Due to the attack method, I created in the warriors file, I have passed in the targets for both warriors. For warrior 1, the target will be warrior 2.

User Feedback

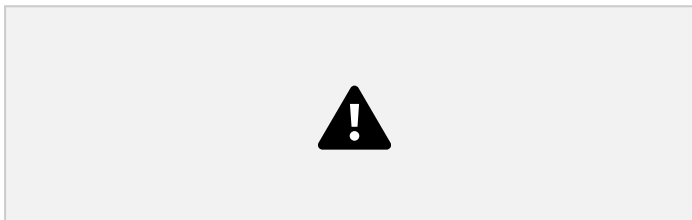
Mateen was pleased that the basics of the attacks had now been built and agreed that the depletion of health should be looked at when creating the health bars as this is logical. Mateen did say however, that there only being one attack made the game rather one dimensional. I explained that at the moment I just wanted to get the fundamental mechanics and collisions working before working on multiple attacks.

Implementing Health Bars

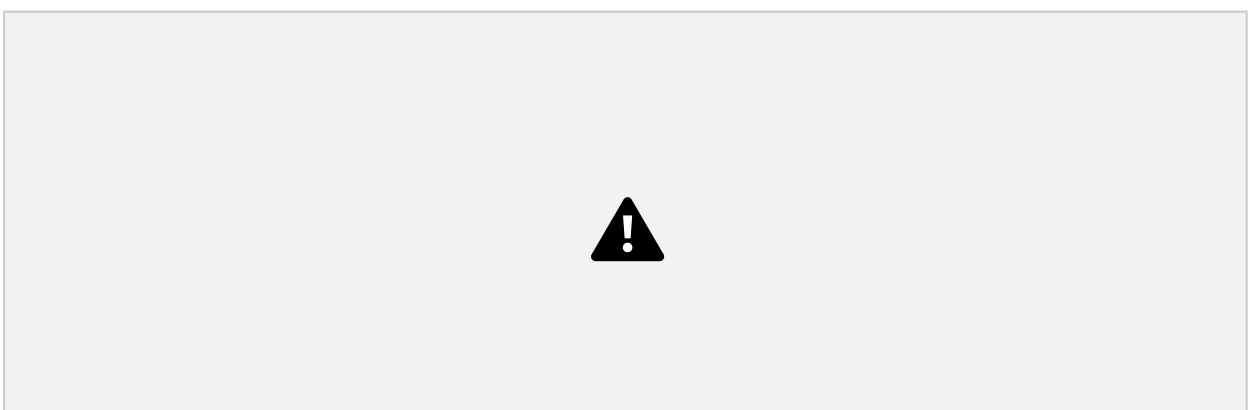
In this stage, I aim to create health bars displayed for either player which deplete upon a receiving a successful attack. I will introduce variables for health and draw additional rectangles on the screen for my health bars.

Iterative development and testing

Made a variable called self. health which is initially set to 100 for both players.



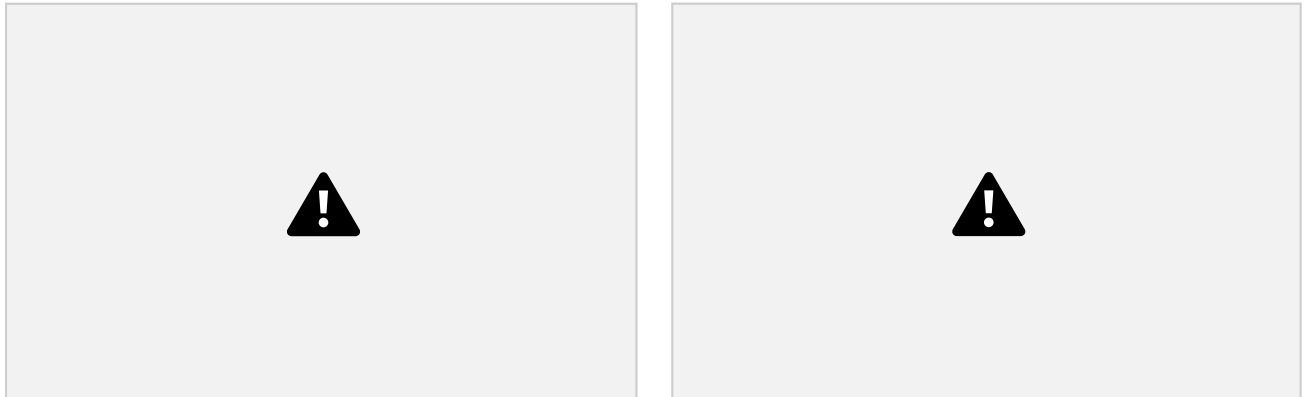
First step, check health variable decreases upon attack. I used a print statement for this, and it was working correctly.



After implementing draw health bar's function.

Two health bars now appear on the screen. I tweaked the height of the bars slightly to make them narrower (25 to 15).

They also deplete. By using a ratio between the health and 100, you can times that by the green bar to show a decrease.

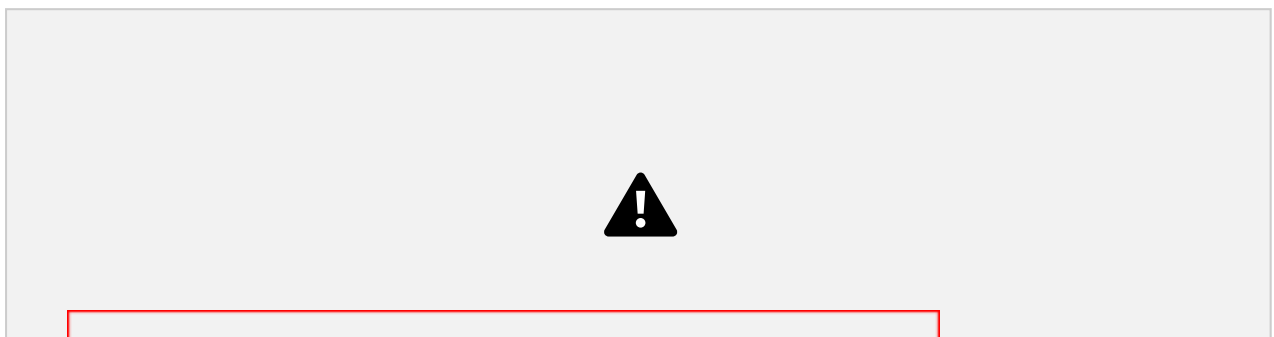


I took inspiration from a YouTube tutorial to code this. They used slightly different sizes and approaches to color, but the logic is similar.

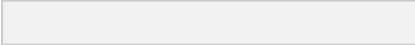
Test Number	What is being tested and relevant inputs	Expected Output	Actual Output
2	Health bars at top left and right corner of screen	Should appear for both players to visually represent decreases in health	Two health bars are drawn to the screen taking both warriors health as arguments.
3	Health bars at start of round	Display only green health level to indicate 100% health.	Full health represented by full green bar.
4	Player 1 Health bar depletion	Depletes upon receiving successful attack from Player 2. Green bar reduces to reveal red.	Player 1's health bar decreases upon receiving an attack from player 2. Depletion is shown using red.
5	Player 2 Health bar depletion	Depletes upon receiving successful attack from Player 1. Green bar reduces to reveal red.	Player 2's health bar decreases upon receiving an attack from player 1. Depletion is shown using red.

Code and Explanation

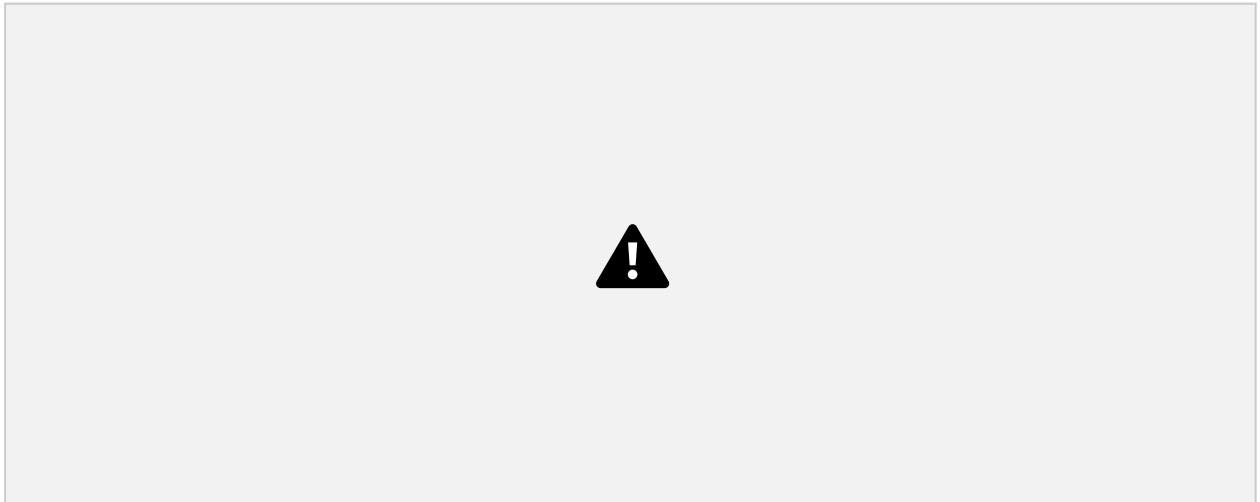
warriors.py



In the defined attack method, I introduced the depletion of health for the target player upon the if statement condition being made. So, if the player attacks and lands successfully, the defined target will lose a set amount of health (-10).

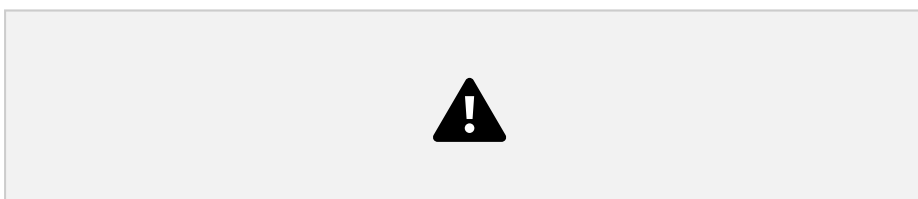
 I defined health and set it as 100 initially for any warrior. This is so I can work with percentage and ratios more easily when implementing health bars.

combatwarrior.py



This function was defined in my main file. There are three draw.rect statements. The first one, draws a Black border of the rectangle which was set after fiddling with the x, y and dimensions. The next one draws the red beneath my green in order to contrast and indicate a health loss. Finally, my green rectangle fits within the black to show remaining health.

The key is my local variable 'ratio', which causes the green rectangle to appear narrower (reduce width) based on ratio of health out of 100. This means that when the player loses say 10 health points, his ratio is 0.9. By multiplying this by the width of the green rectangle, it shows a live depletion. Of course, an argument is passed in when calling this function for both players health (warrior1.health) as can be seen below.



User feedback

My stakeholder and I played a little game and tried to deplete each other's health bar. He suggested that the game was starting to take shape and was happy with the level of health depletion. Although he also said the game lacked any real winner and felt a bit empty without sprites and animations.

This was true, however I assured him that that the scoring and animations will be implemented, and he was overall anticipating my next steps.

(02/03/2024)

Implementing Stamina Bars

My goal with this step is to display stamina bars below the health bars to indicate depletion of stamina every time an attack takes place. Recovery also needs to be shown unlike the health bar. I also want to implement a critical level to stop players spamming attacks.

Iterative development and testing

I tried using a similar logic to the health bars. I set self.stamina to 100. But when I added a recovery function, it just kept recovering beyond 100.



The fix, however, is relatively simple. In my conditional statement, I added an and stamina < 100 and recovery of 1 at a time otherwise, it would keep recovering.

There was a slight issue though, and that is the recovery rate was ridiculously high. This means as soon as the player lost stamina, he gained it. I decided to first put the stamina bars on screen visually to make it easier for me to adjust any recovery rate delays.

I Played around with the y coordinate of the stamina bar to control the spacing between it and the health bar. (went for 54).



I then decided to make the color of the stamina bar slightly lighter.

Now I can visualize the recovery, the stamina bar is depleting but very quickly recovers. I reduced the recovery increment from 1 to 0.05 and the recovery rate is good now. However, when I tested to see what was happening to the stamina, it was a few decimal places above 100 after full recovery.

I then capped the stamina to 100. I also capped it to 0. The following code logic was used:



The values in between, though not accurate to the increment, were not important. Hence, I decided to move on as this was not a major issue.

I decided to add the critical level. This logic simply meant that if the players stamina dropped below 10, the player would not longer be able to attack.



Warrior 1 on the left, cannot attack.



Test Number	What is being tested and relevant inputs	Expected Output	Actual Output
6	When player attacks. Stamina affected.	Value in stamina variable decreases when the player attacks. Increases upon rest.	The stamina variable for each player decreases when attack is made. Increases upon rest until 100.
7	Stamina bars top right and left corner below health	Stamina bars appear below health bar to visually represent increase and decrease of value of stamina.	Stamina bars represent stamina depletion and recovery correctly.
8	Stamina bars depletion	Drains for respective player as they attack. If drops below critical level, player can't attack.	Stamina bar blue level become lower as it drains. If drops below critical level, player cannot attack, and recovery is slower.
9	Stamina bars recovery	Recovers stamina for respective player as they aren't attacking.	Blue level increases to show recovery as player isn't attacking.
14	Stamina bar reaches below critical level	Player with critical stamina should not be able to attack until it recovers stamina.	Player cannot attack until recovered above critical level 10.

Code and Explanation

warriors.py

I defined an attribute stamina in my constructor method and like my health bars, set this as 100 in order to calculate ratios and implement the logic more easily.



The above code is part of the attack method defined in the warrior class. The first if statement essentially causes the attacking warriors stamina to deplete by 10 only if the stamina afterwards is greater than 0. This is so that the player doesn't have negative stamina. The other case, represented by the elif statement, caused the stamina to be set to 0 if the current stamina minus 10 would be less than 0.



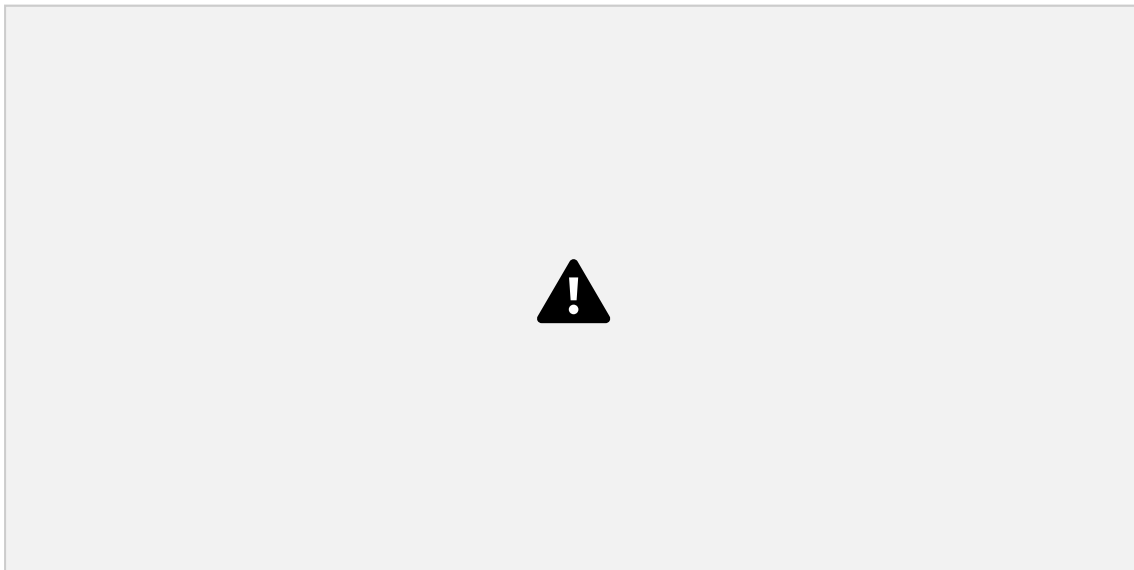
The method recovery handles the recovery of the warrior's stamina and is called for each warrior in the main game loop. The self. Stamina only increases by the increment 0.05 if the warrior is not attacking and his stamina is between 10 and 100. This is because I have chosen a different recovery level if the player drops below the critical level which I have defined as 10. If the player drops below a stamina of 10, his recovery is slowed to 0.01 increase in order to slow him down and teach the user the consequence of spamming attacks too often.

combatwarrior.py

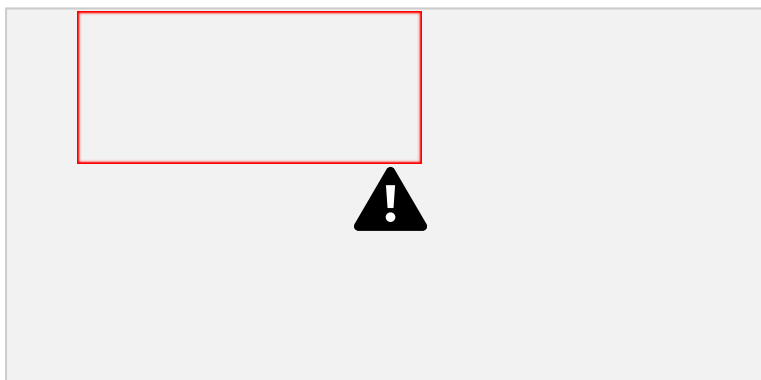


This function is in my main file and draws the stamina bars in a similar manner to my health bars, but recovery logic is displayed. A ratio is calculated and again used to shrink and in the stamina's case, grow the width of the health bar as long as it doesn't exceed 100.

A thin rectangle for the critical level has been drawn at 10% the width of the light blue bar and to the right of its x coordinates.



set another condition in my main game file which only allowed for the players attack to be called if the warriors stamina was above 10. This is part of implementing a critical level, as when they reach this region, they cannot attack until recovered appropriately.



This code draws the stamina bars directly under the health bars and calls the recovery method for both warriors.

User Feedback

Mateen Raza was happy with the stamina bars implementation and when paly testing, though it was a good way to stop spam attacking. However, he did say that he felt the health depleted too much in comparison to stamina and it was still possible if every attack landed to win the round with a constant barrage of consecutive attacks.

I decided to tweak the health variable depletion to 5 instead to fix this. He was happy after I implemented this.

(03/03/2024)

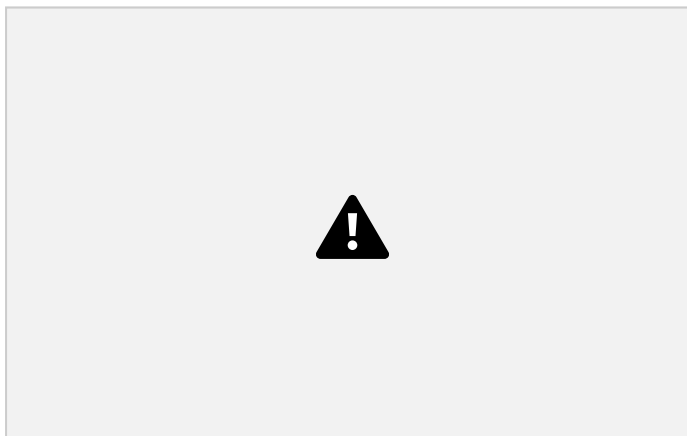
Round Scoring and Timer

For this step, I aim to implement the logic pertaining to round scoring and add indicators to make it clear to the end users. I also will implement a timer count in the round and also an additional way of winning the round.

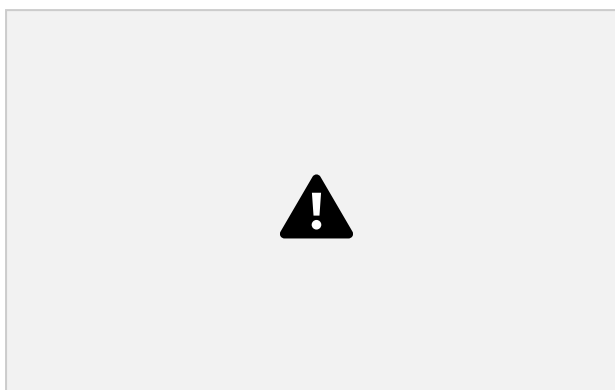
Iterative development and testing

Initially I made the round indicators in size 10 x10, but they were too small. So, I increased to 15 x 15.

I tried implementing the logic. The round indicators successfully changed color, but they changed color for both indicators and, they changed for the warrior who lost the round.



Then I realized that it was a logical error. I had said that if warrior health = 0 change color which is fine. But when drawing them, I passed warrior 1 for their own respective indicators. That means if warrior 1 reached 0 health, his indicator would change. So, I simply switched these around and fixed the error.



After trying multiple times, I decided to give up on having two round indicators. Since my game was best 2 out of 3 rounds, I decided it was best to make it clear of the deciding round and only use one indicator per person.

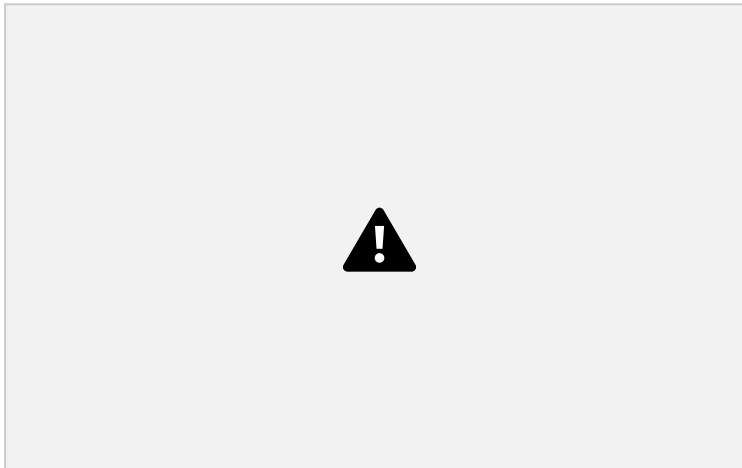


Successfully made round indicator go off for the person that depletes the health bar.

Before implementing a timer, I decided to make the functionality that resets rounds or game over if opponents' health reaches 0. Then when I implement the timer, it will be easier to make another condition for winning the round.

I first started by displaying a message when the warrior wins the round.

I changed the color of font from black to white to make it more visible. I changed the size of font from 30 to 50. I changed the x and y coordinates from (SCRN WIDTH / 2, SCRN HEIGHT/2).



Now a message and round indication to let the player know who won the round.

I tried to implement a function to reset the stamina and health variables to 100. However, this was not displayed on my screen. Then, I realized I was resetting my round continuously. I updated the logic with a conditional statement but still, the health and stamina were not resetting. I then came to realize that I was resetting the round before updating the display, hence nothing was being shown on screen. I then fixed the order of operations error. That still didn't work. I then tweaked some things and managed to get it working. However, the round indicators reset as well. Hence, I updated my round indicators to consider if the round was won or not.

This still didn't work. I decided to test if the health and the round won values were updating correctly. I found out that the health was resetting as it should, but the round indicator was not.



(04/03/2024)

I decided to remove the recently coded bits and rethink the logic. I then managed to get the round_won () to become true when the health reached 0.

The round resets after the end of the round. But the indicators do not update...

The problem was where I was calling my round end function. It was meant to be just before the update.

However, I ran into a bit of an issue. After the round indicator changes, it goes away straight after the reset.

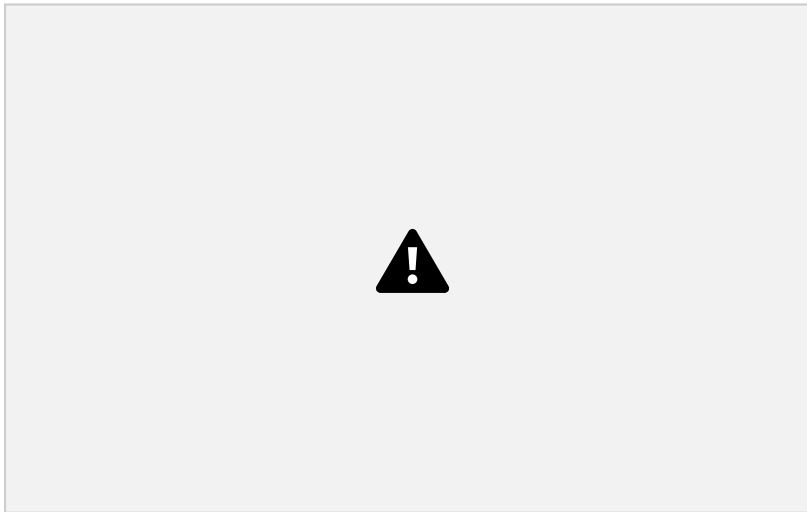
I managed to fix this issue by using a round_result attribute and a string. Now when the player wins a round, the round indicator stays updated.



I have managed to get a message to appear for the respective winner of the round. However, I now want to implement a delay between resetting from round to round. This will allow for the message to be read and gives players time.

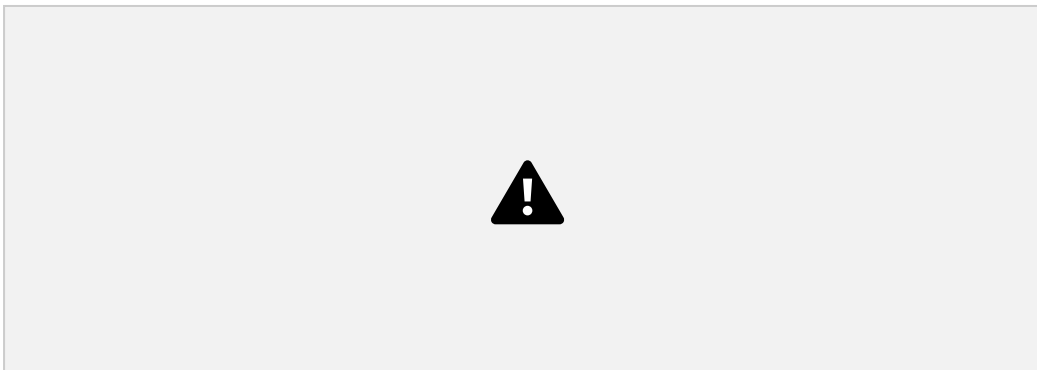
I used the following to achieve a delay to display the message before resetting.



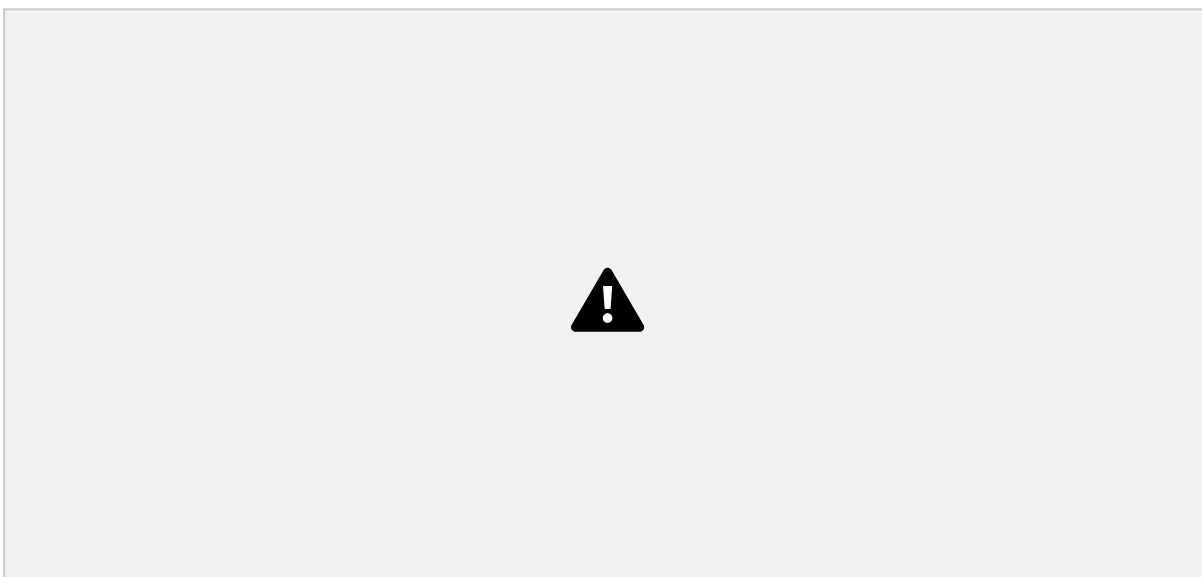


But the delay occurred when the moment the player attack landed, the motion would not be finished before the round ends which would be weird. I decided to move on and focus on the game logic then come back to it later.

Game over after two rounds won. For now, I have made the game over as quitting the game to test that this works.



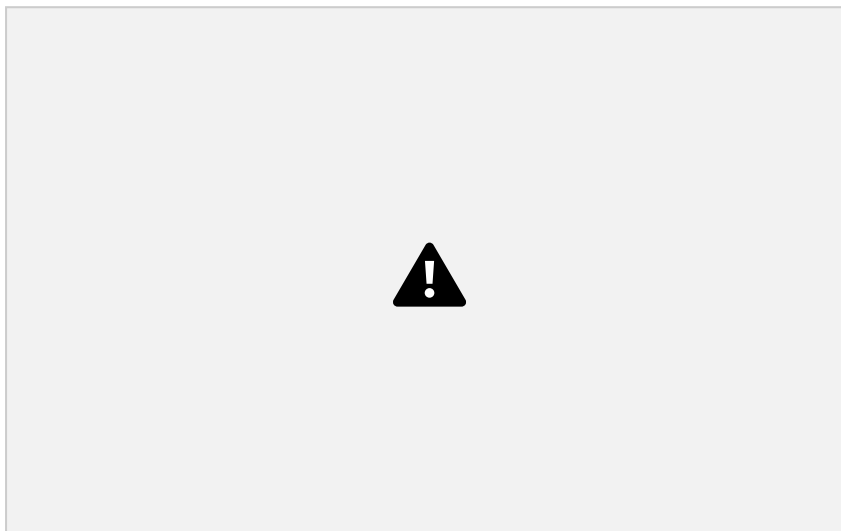
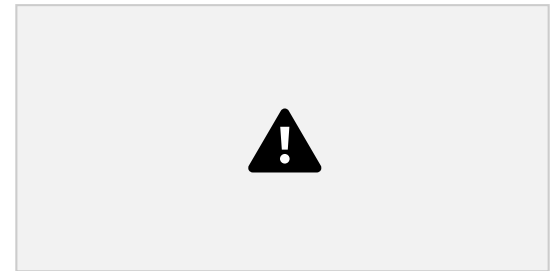
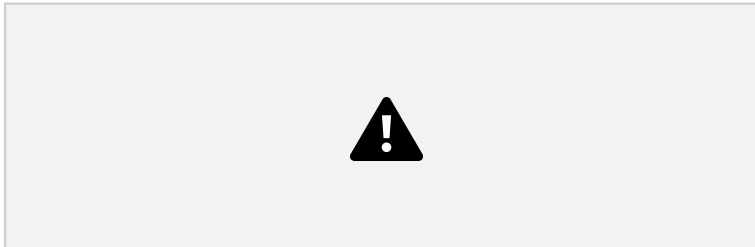
I used a print statement to make sure the rounds counter was functioning as it should.



I noticed a slight problem. When the rounds reset the player positions didn't reset. So, I made sure to pass in an argument for the original x coordinate of both rectangles in the reset status method.

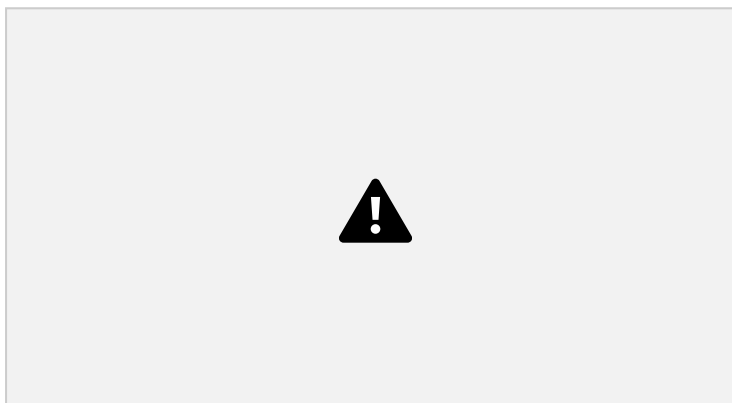
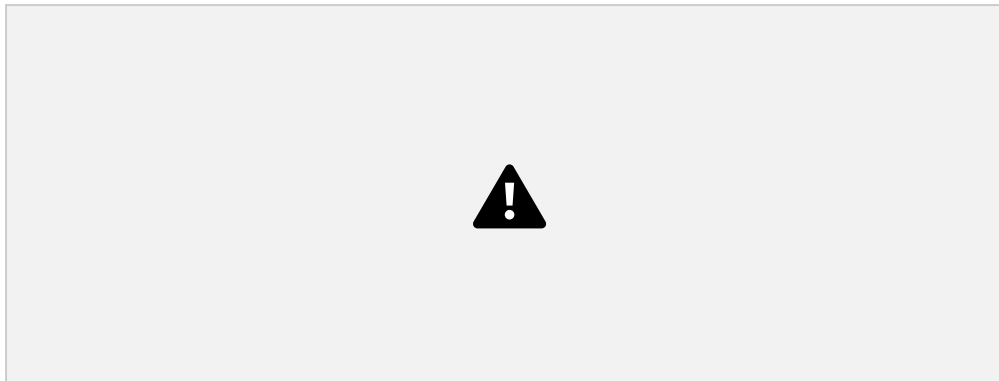


I then added a start countdown using the following logic.

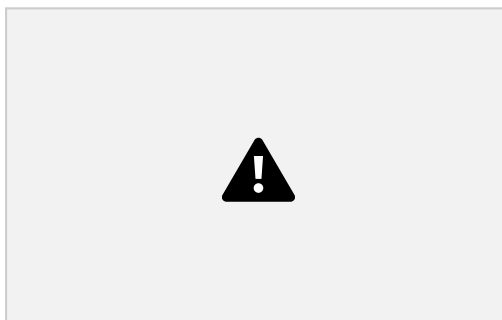


In the final round, I wanted to state, "Sudden death!" in red.

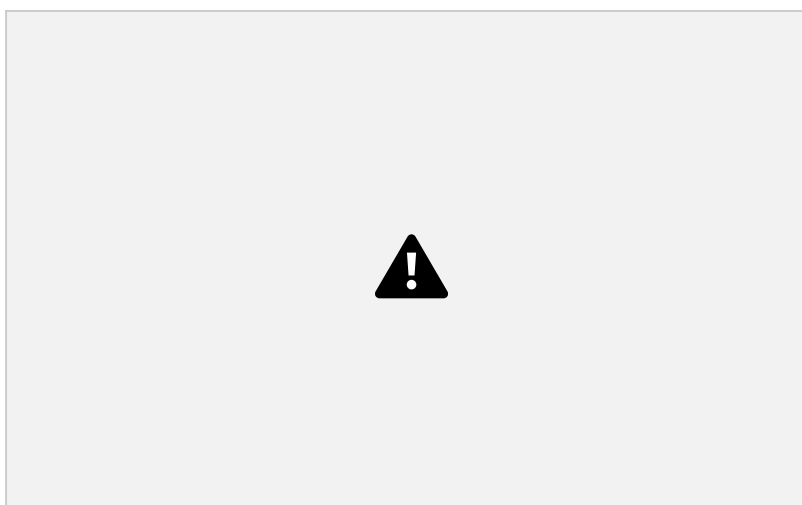
So, I used the draw text function within an else statement, and this was used for final round.



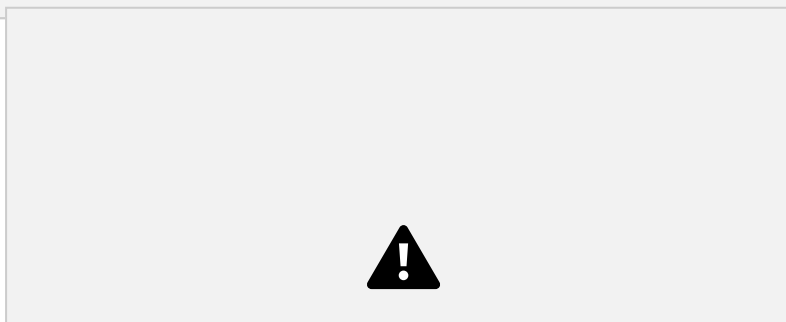
I changed the color of the countdown timer as well for added effect. See below.



Mateen said that the 3! Is not really shown at the beginning of the second and third round. This could be refined.



Now, apart from the streak, my round system and scoring is all working perfectly.



Test Number	What is being tested and relevant inputs	Expected Output	Actual Output
10	Round indicators displayed in top left and top right corner below health and stamina bars	Should fill with a colour based on who won the round. Winner of round gets box filled.	One round indicator per warrior which fills orange based on winner of round.
11	Timer count	Should decrease as fight goes on second by second	*Round timer count decreases during fight. Start timer reduces before beginning of round.
12	Timer reaches 0 (not final round)	Round should end and compare health levels of both warriors. Message to display this visually.	Round comparison occurs if the round timer reaches 0. Message is displayed visually.
13	Health bar completely depleted	Round should end with player with remaining health being declared the winner.	Round ends with player that has depleted the targets health to 0 as the winner.
15	Streak counter displayed above player status bars	Updates once the match is over based on who won. Stored in separate file or database.	This has not been implemented as of yet. I decided that I would do this when trying to implement the leaderboard system.
16	Player wins 2 rounds	Fight is over. Display streaks briefly and visually displays winner of match. Transition to game over state.	Exits the game for now. Game over logic will be implemented later.

Code and Explanation

warriors.py



These are the attributes which I have defined in order to

implement the rounds and scoring logic.



I edited my attack method in order to include a round result and setting round won to true. This occurs when the target health reaches 0. Hence, the warrior who attacked when this occurred will have won the round.



The method 'reset_status' is used after a warrior has won the round. Both players will have their health and stamina set back to 100, and also the warriors x positions on the screen (as y will always stay constant this does not need to be reset) will be put back to the original starting position defined in the main file. Also, the round_won attribute will be set to false when the round resets.

combatwarrior.py



This function draws a small rectangle underneath the health and stamina bars and also updates the round indicators to the color orange if the round result is "won". Otherwise, the round indicator is simply grey. The reason "won" is used is because the round_won is immediately set to false and hence the indicators don't stay updated.



This function is called in the game loop and takes the arguments x, y and the round result pertaining to both warriors. So, if warrior 1 wins the round, the round indicator is updated accordingly.



The code to my left is defining variables.

Can_attack is set to false by default until we decide to set it to true and essentially is used to control when the players are allowed to attack .

The other 2 relate the initial start and round countdown. Last_count is a variable which using

the Pygame.time.get_ticks() function, represents the recent count or tick.



The “round_end” function takes the argument winner name which is called for both warriors in the game loop based on certain conditions. It uses the global variable round_countdown and resets this countdown to 30 after the round has ended. The rounds counter defined in the warrior file is updated (as this is used to keep track of the winner overall) and text is displayed using the draw_text function. I have added a slight delay before resetting both warriors’ status as this should not happen immediately for smoothness and logic purposes.



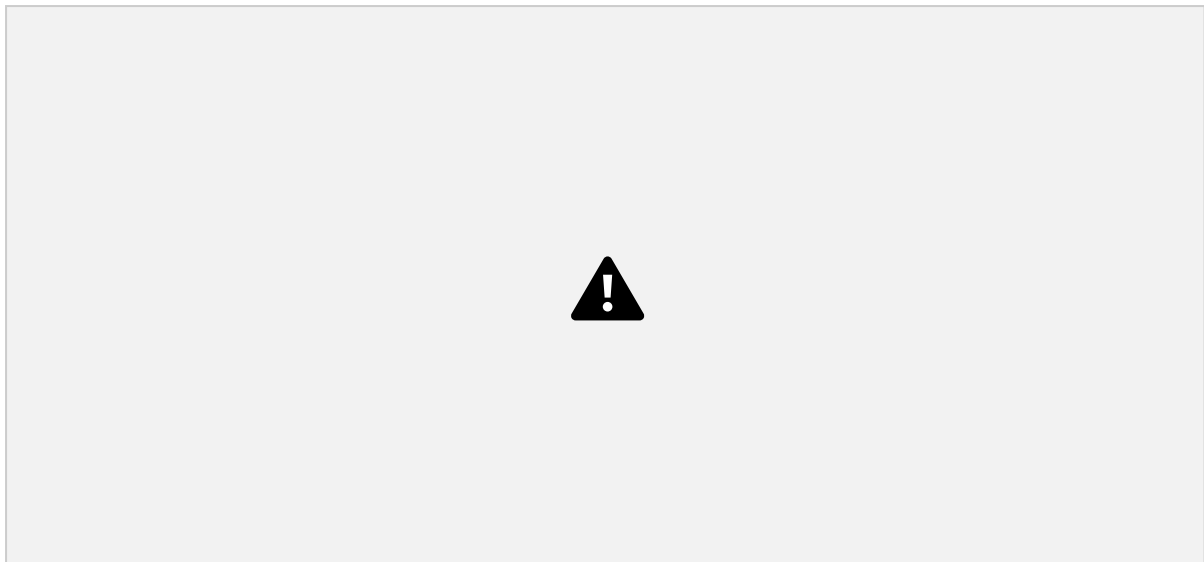
However, there is also a function which defines what occurs if the timer runs out. It compares both warriors' health and based on this sets the round_won to True and round_result to "won" as this is used to keep the indicators updated.



Above is the logic for the round over. At the moment, I quit the game upon game over as I will add this state later. The round_end function is called is either warrior won on health depletion, and the round_end_ontime is called if the round timer reaches 0.

I have indented all the code related to actions while the warriors are fighting within an if statement, which only allows these to take place if the start countdown has reached 0.

The round timer is drawn on to the screen and if the count timer – last_count is greater than 1000 milliseconds or 1 second, the round countdown being displayed is updates as the value reduces by -1. This gives the impression of a timer countdown.



If the start countdown is greater than 0, text is displayed to the user. It is different if the warriors have both reached 1 round each and the SUDDEN DEATH message is displayed. A similar logic to the round timer is used for the start countdown.

User feedback

My user said that he was pleased that the game mechanics and round system were all working as they should. He suggested that implementing the other game states and animations would complete the game.

Mateen did suggest that the rounds were too short however, and that it did not give enough time for either player to deplete the other players health bar. I said that for now this was not a concern and could be focused on post development. Mateen agreed with my decision to move forward with the development.

Mateen also pointed out that the 3 in my round countdown isn't really displayed properly and is barely noticeable which takes away slightly from the count.

(05/03/2024)

Milestone 3: Paused

The goal of this milestone is creating a separate paused game state which displays a paused menu when the correct button is pressed.

Iterative development and testing

I decided to set my game states using variables.

Paused = False. On key press this will be set to true, and I can implement the functionality.

I created a little paused menu. So far, it only has text giving instructions. If you push anything, nothing happens. I changed the position of this text later after implementing this logic.

I decided not to use buttons as I felt that I would be constrained with time, and it would not be a good idea to implement additional functionality such as buttons.



Now, I added the logic to un-pause the game.



However, when I tried to un-pause, nothing happened.



I then changed it so the game would check for key presses in the paused state where I was displaying the text.

It did un-pause, however, the second the space bar is pressed it resumed. I think this is because in my Pygame.KEYDOWN statement, I am also not checking if paused is true.



It still didn't work as expected. So, I tried dedenting it to see if it would change anything.



Still didn't work. Then I tried to use another button other than SPACE to see if the error still occurred.



When this didn't work, and the paused menu disappeared straight away on pause, I had reconsidered the game logic and use a trace table to test what was happening.



The paused and resume function now work.

I added the logic for escaping.



Now, when the player pressed escape in the paused state, the game quits.



I adjusted the message displayed and added a message above the options to indicate paused.

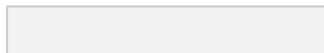
Test Number	What is being tested and relevant inputs	Expected Output	Actual Output
1	User input causing the playing game state to be paused	Pause any loops currently being run for the playing state on key press	Sets the paused loop to true and the playing state no longer runs until resumed.
2	Paused menu	When paused, a menu is displayed with options to resume or quit.	Options as instruction of how to quit and resume are displayed.
3	When user resumes. (from paused menu)	The paused loops should now be resumed, and the game state should now be back to the playing state.	The paused state is set back to false and the game runs from where it left of.
4	When user quits. (from paused menu)	The game state should transition from paused menu to the main menu. Playing data is saved.	The game quits upon esc being pressed. Playing data is not saved.

I did not implement the functionality to quit to main menu from the paused menu due to me not developing the main menu just yet.

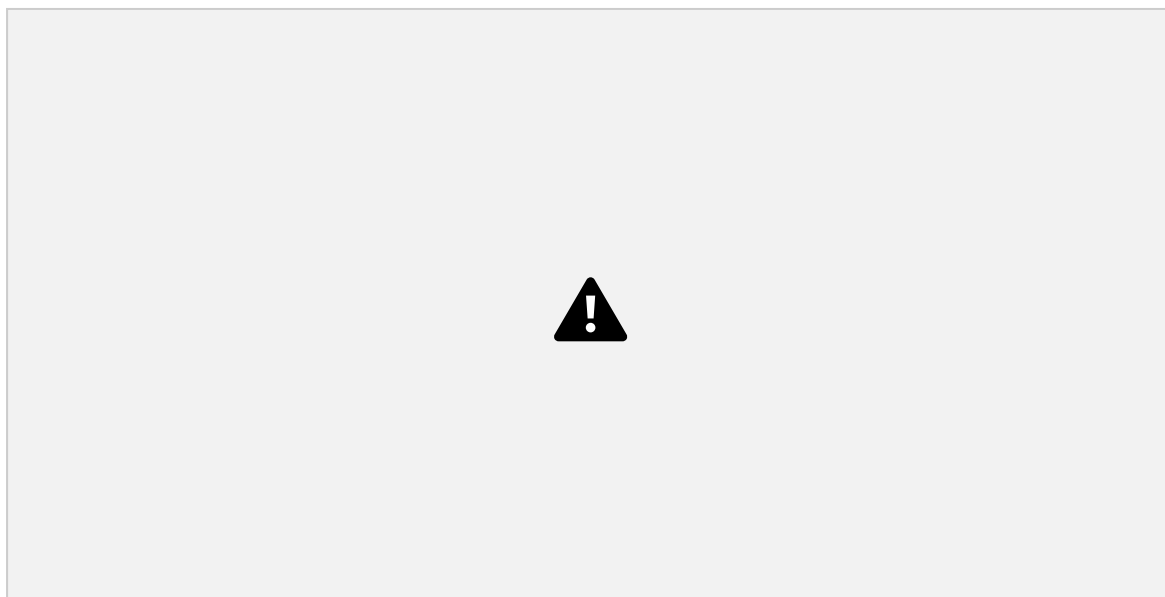
I also decided after implementing the functionality to create the pause logic in a separate function within the combat warrior file. This was done to make the code more readable and maintainable.

Code and Explanation

combatwarrior.py



This is the paused variable which is initially set to false. It is only set to true if the paused logic is called.



This function “pause_logic” is called when a key down event occurs in Pygame. Since paused is declared outside of any function, I have to declare this in my function. I used an if statement which allows the user to use the space bar to pause and resume the game as this is more intuitive. This only has one condition, and that is if the space bar is pressed. In my elif statement, this is a functionality of the paused menu which allows the user to quit the game only if pressed while the paused variable is true.



Within my main game loop, if a key down occurs, the pause logic may be called if the key pressed was space (hence the condition within my function).



This function simply handles what is displayed on the paused menu and uses the draw text function of Pygame. It takes the parameters of text, the font type which I defined within my program, the color, the x, and y coordinates of where I want this text displayed. The text is used to instruct the user of what keys to press to perform a certain action on the paused menu. This is to make up for the lack of buttons.



This statement is in my game loop and calls the function to display the relevant text when paused is true.

User Feedback

Mateen was ok with me using keys rather than buttons. To summarise his words: it would be better to focus on the more important aspects such as game states and additional functionality rather than visual things such as buttons. But he did feel that this didn't give the game as high quality of a feel, which is an issue which will be discussed in the evaluation.

(06/03/2024)

NOTE:

Before I develop the next milestone, I decided to work on the game over state first before looking at the other milestones. This is due to me already having implemented a small functionality for game over – which currently quits the game when game is over – and I thought it would be better to implement this milestone before looking at the others.

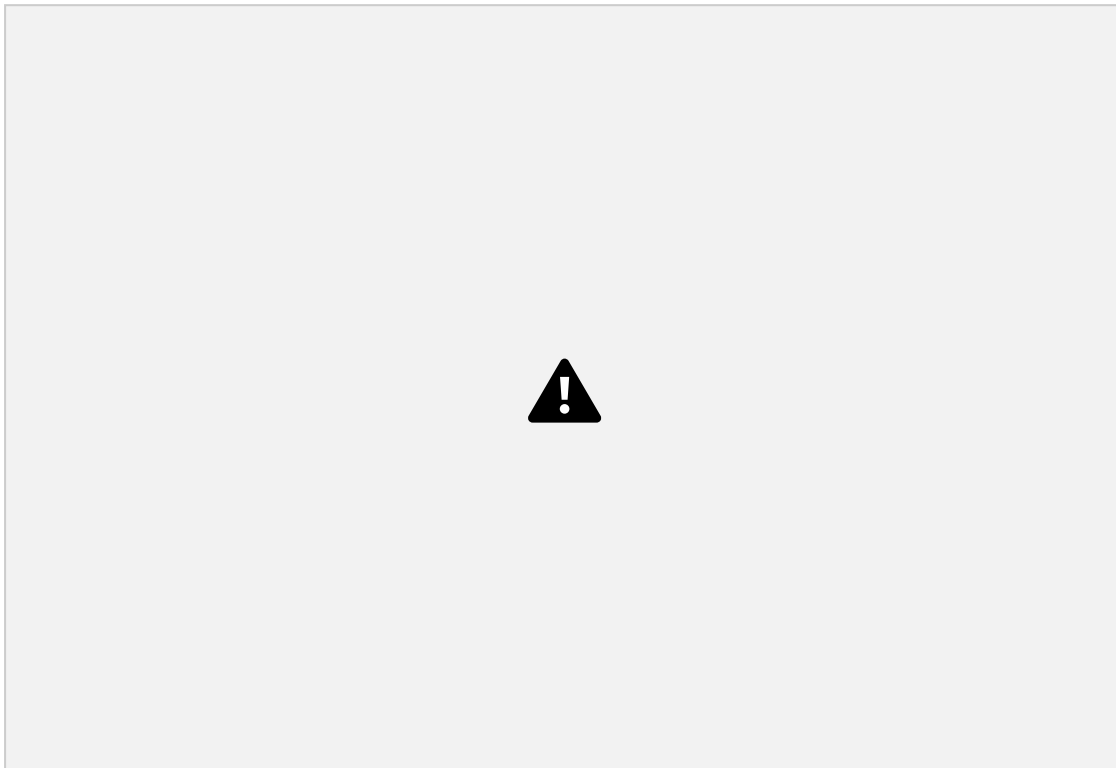
Moreover, since one of the functionality of my game over state is to quit to the main menu or quit the game, I decided to combine these processes since it uses a similar logic to create the game state.

***Milestone 7: Game Over interface + Milestone 4: Menu interface**

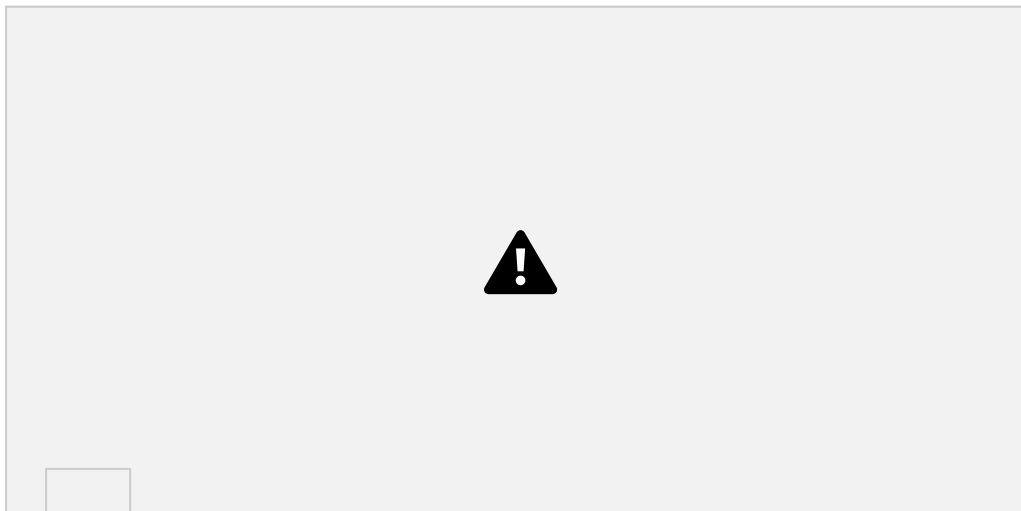
For this milestone, I aim to have a game over screen with functionality to quit to main menu or quit the game, as well as develop a main menu screen which is displayed at the start of the game.

Iterative development and testing

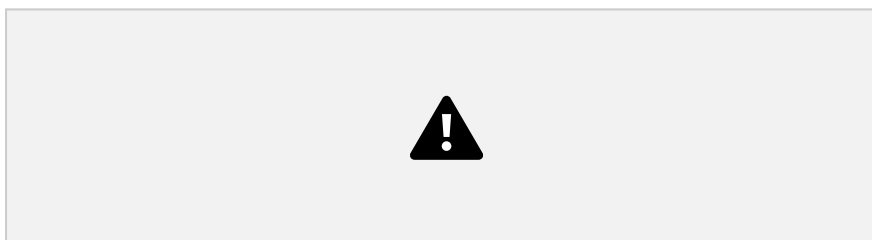
The game over logic was working fine as long as both players won a round. However, I tested the game when one of the players won, and the game over state and the round reset occurred at the same time. This was because my function initially stated that if warrior 1 rounds counter + warrior 2 rounds counter = 2, then enter the sudden death round. However, this means if warrior 1s counter was 2, the sudden death round would initiate! Hence, I corrected it and made the condition more specific.



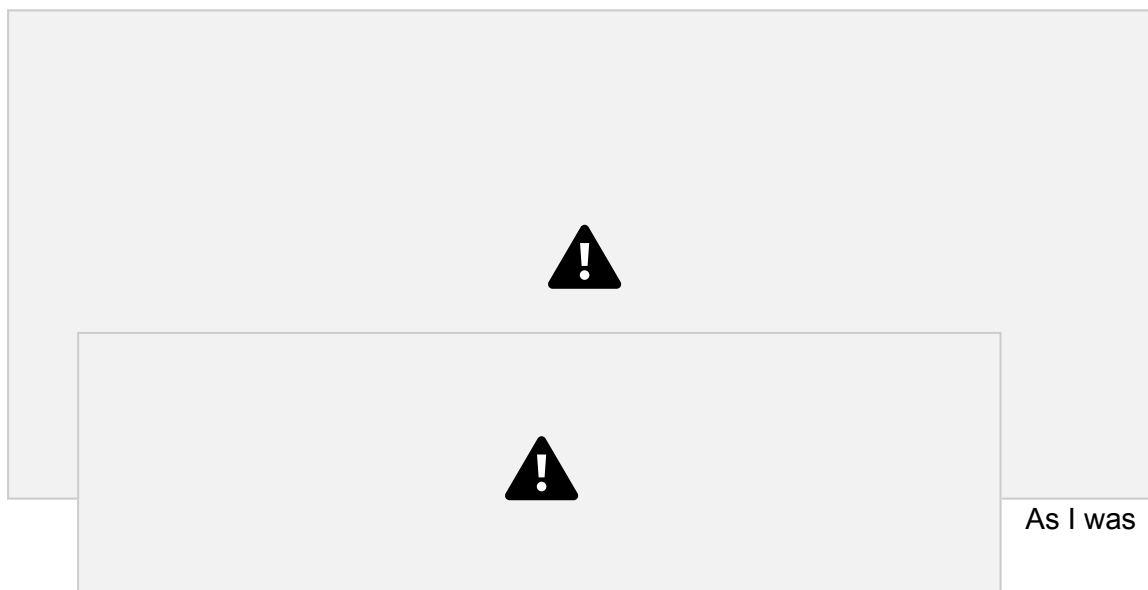
also used a print statement to keep track of either warriors round counter to make sure that the game logic wasn't perverse due to incorrectly updating or broken variables. However, the round counters were fine, and the above fix corrected the error.



However, I wanted the warriors, health bars and round indicators to disappear on the game over state.



Then I used the following logic to set game over to true. And when this occurred, only the game over and the winner would be displayed.

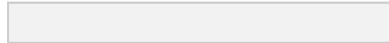


As I was

implementing the game logic, I noticed that on the countdown to rounds other than round 1, the player was able to attack. The problem occurred because they can attack variable was

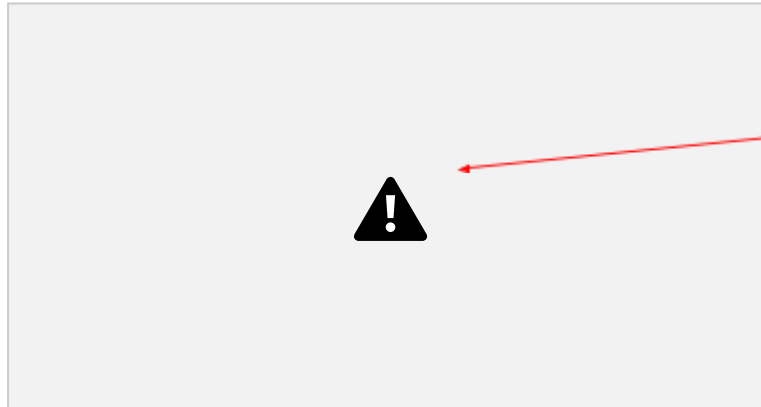
being set to true initially, but never reset again. I fixed this by setting can attack to false whenever the start countdown was greater than 0.

I decided to implement my menu interface. As this took a similar logic to developing my other game states, I will simply do the same process again. However, this time, the main menu will need to be initially set to True as this is the first thing that will be seen.



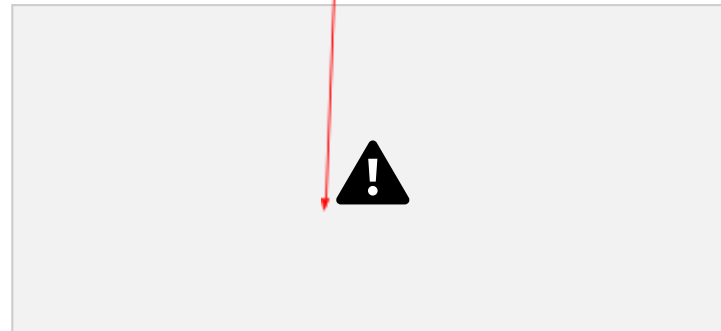
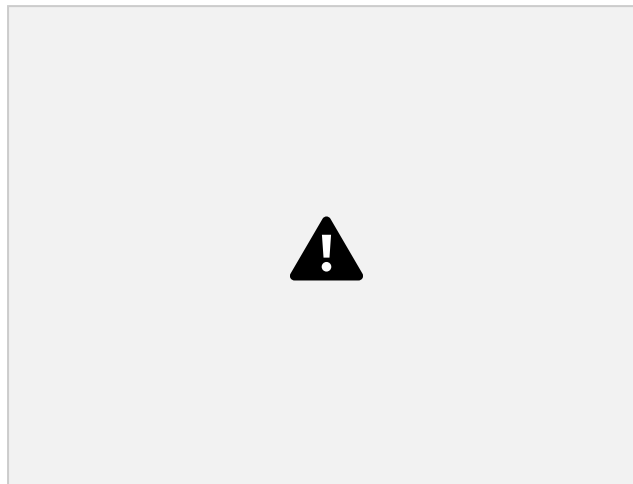
This has been done and I will now work on the logic.

I started by using an if statement to control what happens when the main menu is true.



I also added a key to view the leaderboard as this is part of my functionality for the main menu. For now, I added a pass as I haven't yet worked on the functionality for this.

The escape button allows the player to quit the game.



The r button sets the main menu to false. This allows for the playing game state to occur.

I will define the function to draw the text for the main menu and test what happens when the logic is implemented within the main game loop.



This for now displayed a blank screen with my game background. I then played around with the x and y coordinates to get the text displayed in a suitable position.



(07/03/2024)

I decided to put the game over logic within a separate function as well.



I ran into issues in the buttons or key presses working for the game over or main menu. Then I realized that the event keys were only being checked to see if can attack was true. So, I nested all those with only key down being a condition and not relying on being mid round which the game states game over and main menu are not linked to.

Then I ran into another error, when I tried to play again, after quitting the main menu, the game would not reset. I resolved this by making rounds of both warriors to 0 after game over.



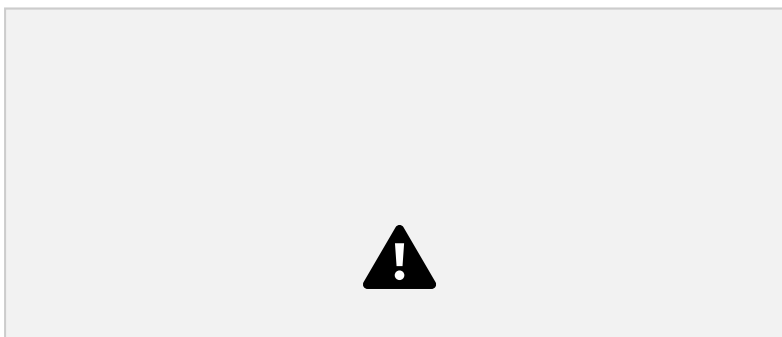
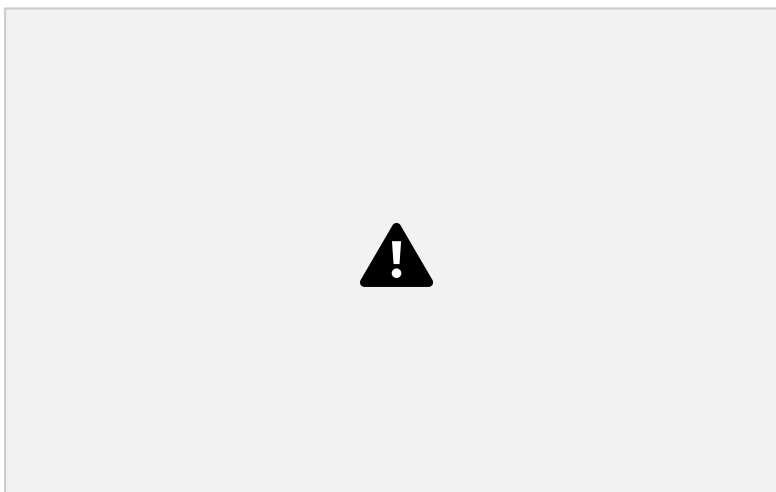
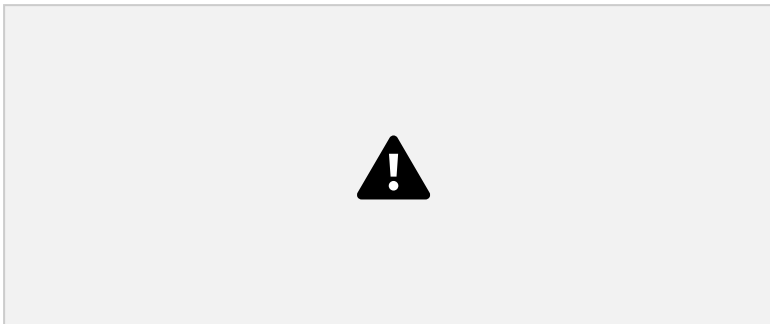
Everything was working fine now – however, the round indicator of warrior 1 was not resetting. This was because the round result was still being saved from the last game. I wanted this not to be the case. Hence, if main menu and r were pressed, the I reset round results. This solved the issue of the round indicators.



This fixed the issue. However, I then ran into another problem, on the game over state, if m was pressed, it would not escape to main menu.

I managed to fix this by setting the game over to false when m was pressed.

Then I ran into another problem, the game wasn't pausing mid round. So, I called the pause logic when key down and can attack is true (which basically means being mid round).

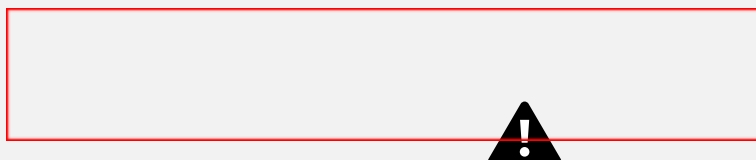


Test Number	What is being tested and relevant inputs	Expected Output	Actual Output
1	Fight has ended	Game state should transition to game over state	Game over menu appears when fight has ended
2	Upon game over transition	Game over menu with 3 on screen buttons new match, rematch or save and quit.	No buttons used. Rematch has not been implemented as nothing is being stored and there are only two warriors for now.

Test Number	What is being tested and relevant inputs	Expected Output	Actual Output
1	The background for the menu screen appears. Game transition from loading screen.	The background for the menu screen fills up the entire interface.	Background is the same for the menu as for the fighting.
2	Buttons displayed correctly and in order.	Buttons appear on the menu.	No buttons are displayed and instead prompts for key presses.
3	Mouse clicks on the GUI buttons.	Transition to the selected game state with relevant loading screens.	Transitions to playing state. No loading screens or mouse clicks used.

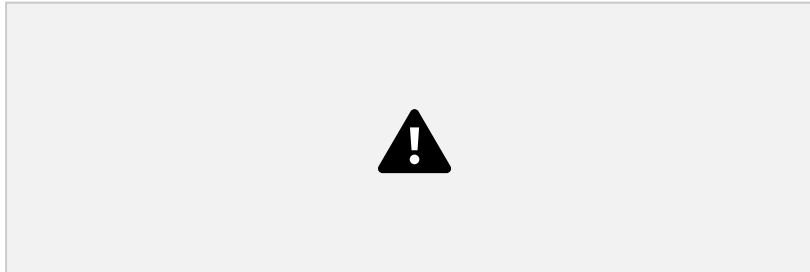
Code and Explanation

combatwarrior.py



In the code above, there are two functions. One which displays the text on the main menu screen, and the other which displays text on the game over screen. Both use several draw text functions as the paused screen did. However, for the game over screen, based on who won the fight, a message stating who won is displayed.

Below is in the main game loop, which calls the relevant screens is the state is true.



In my `game_over_logic` function, I've implemented the functionality which allows the user to quit the game or return to main menu. This is done through the `event.key` function. If "m" is pressed, the game over state is set to false, and the main menu becomes true again.

Above, the key checks only occur if game over is true, which is based on the following conditions:



To the left, the menu logic function resets the rounds counter and results of either warrior to 0 or None. This is done so that the game does not pickup where it was left after the game is

over. The exact same logic for quitting the game is

used as in the game over function, however as main menu is initially set true, it is set to false when the player presses r (i.e. starts the game.). This was done because the main menu needed to be the first thing displayed when the game was opened.

User Feedback

Mateen was pleased with the game states paused, main menu and game over being implemented and that something happens at each stage. However, Mateen said that he felt the transitions between game states were not very smooth and that it wasn't graphically pleasing.

Due to implementing very advanced graphics in Pygame being tricky, I was not able to do anything to fix this issue. I said I can only move on with the final areas of development. My user agreed that moving forward was the best thing to do and these issues could be worked on later.

(08/03/2024)

NOTE: Change to development.

Unfortunately, after considering everything, I have decided that I will not go ahead with implementing the warrior and map selection interface. This is because I feel that I am under time pressure and do not have enough time or knowledge to implement such an advanced graphical interface with Pygame.

I consulted my user, Mateen, about this and he was rather optimistic stating that the only thing to do was to work on other areas of the development and perhaps come back to this stage once I was comfortable with coding this.

Test Number	What is being tested and relevant inputs	Expected Output	Actual Output
1	Box prompt for entry of user surname before warrior selection.	Box appears in centre of screen prompting user to enter username.	N/A
2	Keyboard presses to enter username in box prompt.	Key presses will display respective characters.	N/A
3	Box prompt disappears upon user clicking enter key (if enough characters)	Box prompt disappears and player 2 is presented with box prompt.	N/A
4	Box prompt does not disappear until valid username inputted including at least 4 characters and not already taken.	Box prompt stays until valid username input.	N/A
5	Database registers valid entered username.	(back end) username added to database.	N/A
6	Screen is split into two sides for player 1 and 2 on GUI.	Player 1 side on left, player 2 side on right, indicated by markers for player 1 and 2.	N/A
7	Character interface displayed and mouse hovers on warrior images.	Animation to show selection, character	N/A

		summary and attributes appear.	
8	Mouse clicks on warrior images	Sound effect and cue as confirmation.	N/A
9	Both warriors selected and confirmed.	Transition to map selection.	N/A
10	Mouse clicks on map images.	Sound effect and animation. Transition to playing state.	N/A

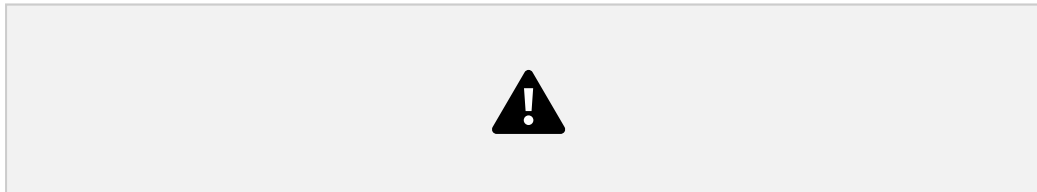
Milestone 6: Sound effects and background music

For this milestone, I aim to add sound effects for the warrior attacks and movement and background music throughout the game.

Iterative development and testing

When I added sound effects initially, it was working. But when I added background music, it seemed to be interfering with the sound. This seemed to be because I was using the same function (`pygame.mixer.Sound`) for both the music and the attack sound.

Hence, I changed the code from:



To:



I used the other `Pygame.mixer` function for music.

I thought that the music playing in the background was far too loud. Hence, I used the `set volume` function. Moreover, I noticed that the music stopped playing once the track had ended. I found in the Pygame documentation that I could loop the music by using the `.play(-1)` feature.

**(09/03/2024)**

I felt that adding additional sound and intensifying the music was an added feature that was not necessary to the overall development of the game. This would require me to gain more knowledge of the Pygame mixer and also to edit the background track I had added to make a more intensified copy. So, I decided that for now, the sound effects and music were adequate.

Test Number	What is being tested and relevant inputs	Expected Output	Actual Output
1	Background music for main menu state	Background music should play when player is navigating menu and selection.	Background music plays when player is navigating menu. Selection not applicable.
2	Background music in the rounds	When entering the playing state intense background music should be played.	Background music stays the same.
3	Background music in the final round	Music should intensify or change	Music Does not change or intensify.
4	Upon attack by either player	Sound effect upon attack depending on the warrior	Sound effects have been added for attacking and jumping.
5	On screen button press	Sound effect upon presses of these buttons	N/A.

Code and Explanation

combatwarrior.py



I have defined two variables, `attack_sound` and `jump_sound` which use the `Pygame.mixer.Sound` function in order to load the `.wav` files for both effects. This has been done so I could easily call them in later and alter if needed.

The other 3 lines relate to the background music. Using the `Pygame mixer`, I loaded the relevant background music file and set the volume to half the original due to it being too.

The `pygame.mixer.music.play(-1)` simply makes the background music loop when finished.



Above, is the attack sound and jump sound being played upon the jump happening. I simply called the `.play()` function to make this work within the game.

User Feedback

My user felt that the sound added an extra layer to the game, and also provided an audio representation of an attack. He felt this helped the gameplay more than just being superficial.

Mateen did suggest that the background music was repetitive and could be intensified or changed at certain points.

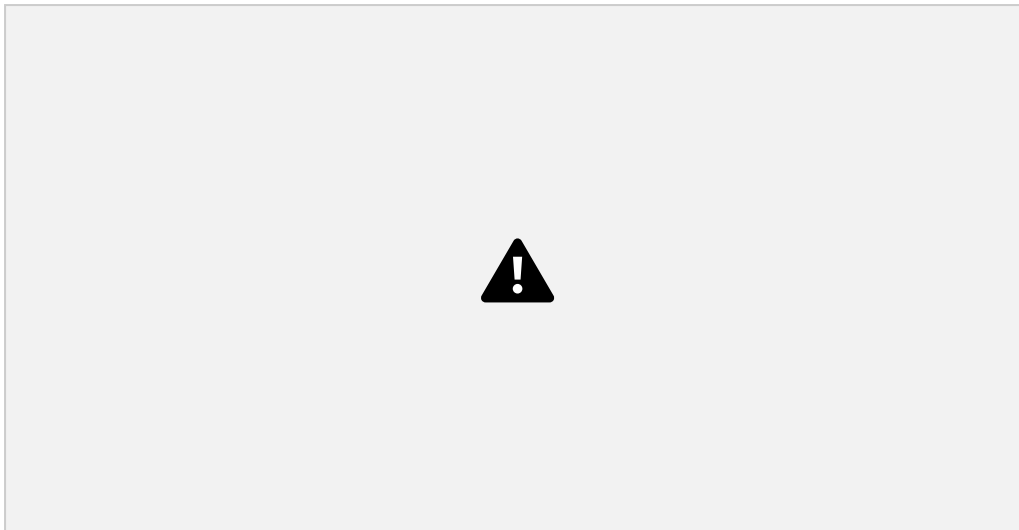
(10/03/2024)

Milestone 8: Leaderboard display

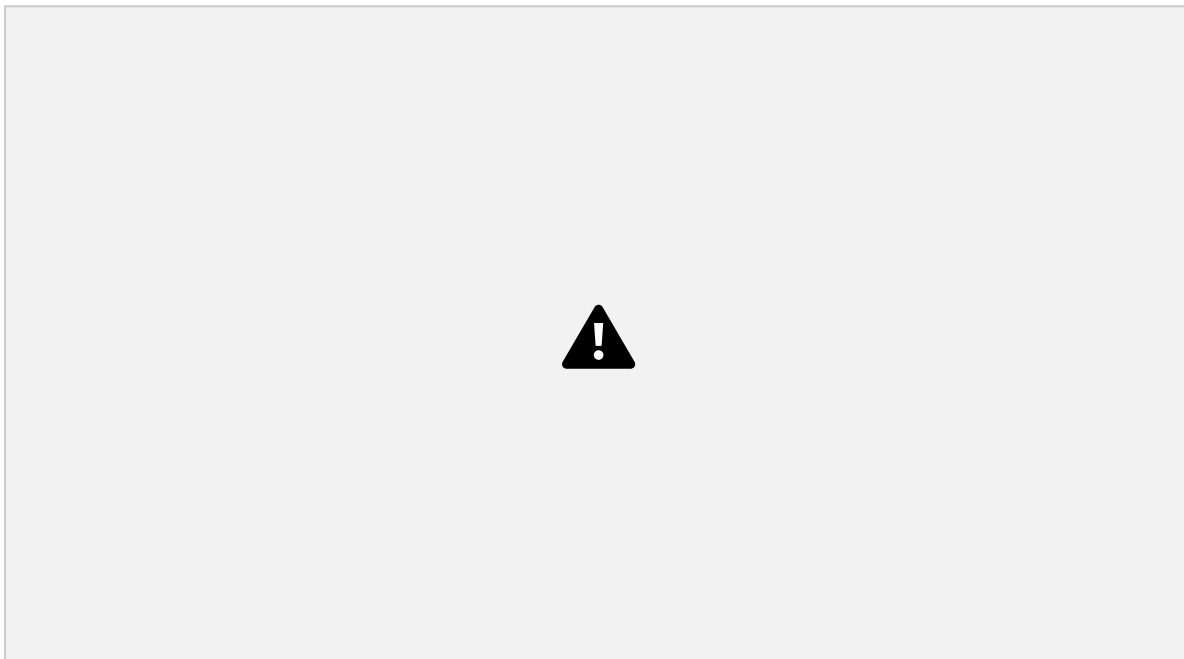
With this milestone, I will attempt to implement a leaderboard within my game. This will require the use of an external database and some query language to communicate with it in `Pygame`.

Iterative development and testing

To implement my leaderboard, I decided to use `SQLite` as my database engine. This is because it is free and seems to be the least complex to implement in `python`.



I set it up so I can give it commands from my command prompt and create the leaderboard database.



By simply entering the mode to control the database engine, I entered a filename for the leaderboard and created it. I used the `.databases` function to display this file in my file explorer.

I successfully created the database. Now I can connect to it in my Python file and create a table within it.

I have defined two functions, one to update the leaderboard, the other to display it. I have largely taken inspiration from tutorials to create both functions as I have no prior experience in connecting to a database using Pygame. However, I made sure to understand what I was implementing and both function require a connection to the database and to close this connection when called.



I have defined two new attributes in the warrior class. Username is used so that the leaderboard can attribute a streak to a particular user otherwise it would be a list of values.

I also added the streak attribute which will be displayed next the username.



Before implementing a screen and separate game state to enter the username, I decided to do this through the terminal at first to check that everything was working properly.



Then I realized I had already created a table with the same name, so I commented this command out to test it.



Then the game ran and took both usernames:



However, nothing was displayed on the leaderboard yet. This was because I was not calling either function.



I did this and received the following error. I had to change the command I commented out to an ALTER command to add the columns to my table.

This did not bring me any results. I looked closely and realized that I was creating a table and another table again and again. This is because the python command would create a table and the table would not disappear upon editing the command. Hence, I ran into problems. So, I ran the create table command once and commented it out as it was no longer necessary.

This worked but when I pressed v, the names were all being displayed on top of each other and only momentarily.

To fix them being displayed on top of one another, I created a y offset which increased every time so the names would be displayed below each other. I tested this and now the users and the streaks were being displayed below one another.



However, there were too many on one screen. Hence, I decided to limit it to the top 5



players.

This did work and now I wanted to keep the names displayed until the user decided otherwise.

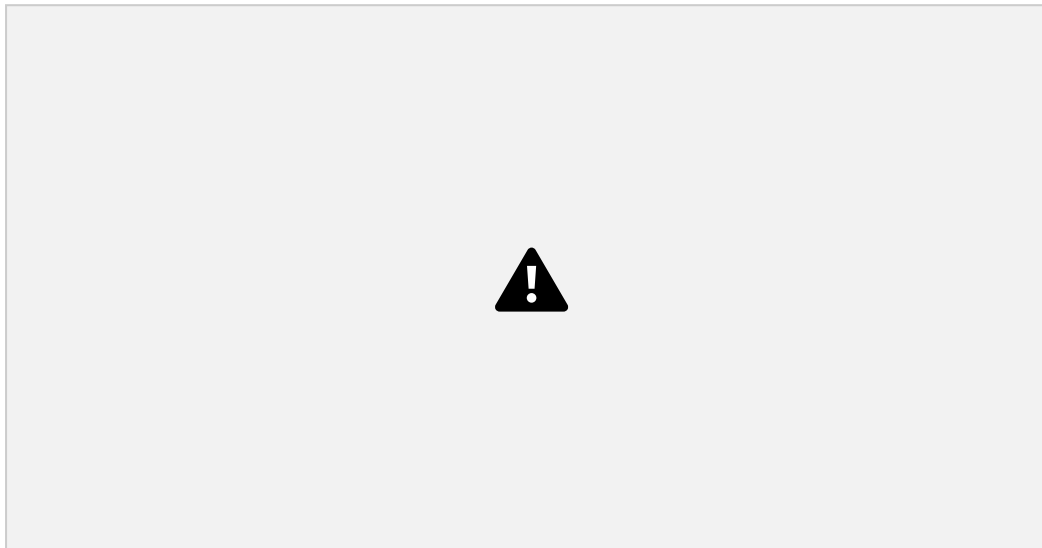
I tried to do this through a while loop, but this caused an issue.



Hence, I decided to remove that while loop and run it again to make sure the names were being displayed albeit momentarily. They were.

I eventually decided after trying a few things, I decided not to pursue displaying the leaderboard as I ran out of time. I decided this would be a post development issue and something which can be expanded upon in the future.

I commented on the button which is pressed to go to the leaderboard as this was causing the program to not respond.

**NOTE (11/03/2023)**

Unfortunately, after trying to get the Database to work and display, although something seemed to be registering in the database, it was not displaying correctly within the game. I felt that I was running out of time and that I could not implement this feature. This is something I will be discussing later.

My stakeholder didn't have input in this decision of course, however he understood the reason I decided to stop the development of the leaderboard.

I have explained what I tried to do with the code and everything I have done so far has been listed regarding the database. Hence, there will be no explanation section as I was not able to complete this myself.

Evaluation

Evidence for Testing and Function

For this section, I have decided to use a video to evidence the testing and function of my game as it is a much clearer way of demonstrating each test and I can reference these more

easily. Hence, each section of the tables below will include timestamps for relevant sections of the video.

For the purposes of the video, I have increased the round timer to 200 in order to provide the time necessary to demonstrate features, test conditions etc.

Milestone 1: Warrior Movement and Attacks

Test Number	What is being tested and relevant inputs?	Expected Output	Timestamp	Pass/Fail
1	Game window	Suitable Game window appears with background image.	0:07	Pass
2	Instances of warriors	Two distinguishable warriors appear on screen.	0:32	Pass
3	Warrior horizontal movement with keys a, d and RIGHT, LEFT	Both warriors move left and right on key press.	0:35	Pass
4	Warrior stays within bounds of the screen (horizontal)	Warrior does not move of the screen when it is moved to the edge.	1:05	Pass
5	Warrior vertical movement with w and UP key.	Warrior jumps. Displaces from origin in minus y direction and returns to 0 relative to floor level.	1:45	Pass
6	Warrior stays within bounds of screen vertical and does not go below the floor	When jumping, warrior does not drop below floor level.	2:29	Pass
7	Warrior attack. Input keys controlling each player	Warrior performs an attack on key press.	2:50	Pass
9	Sprite sheet for animation of attack, movements (MAY BE IMPLEMENTED AFTER FIGHTING AND ROUNDS)	Extracted frames repeatedly looping cause illusion of smooth movement	3:03	Fail (N/A)

Milestone 2: Playing- Fighting and Round system

Test Number	What is being tested and relevant inputs	Expected Output	Timestamp	Pass/Fail
1	When player is in range, attack registered	Health variables depletes to indicate decrease in health	3:18	Pass
2	Health bars at top left and right corner of screen	Should appear for both players to visually represent decreases in health	3:18	Pass
3	Health bars at start of round	Display only green health level to indicate 100% health.	0:32	Pass

4	Player 1 Health bar depletion	Depletes upon receiving successful attack from Player 2. Green bar reduces to reveal red.	3:39	Pass
5	Player 2 Health bar depletion	Depletes upon receiving successful attack from Player 1. Green bar reduces to reveal red.	3:20	Pass
6	When player attacks. Stamina affected.	Value in stamina variable decreases when the player attacks. Increases upon rest.	5:01	Pass
7	Stamina bars top right and left corner below health	Stamina bars appear below health bar to visually represent increase and decrease of value of stamina.	3:18	Pass
8	Stamina bars depletion	Drains for respective player as they attack. If drops below critical level, player can't attack.	5:02	Pass
9	Stamina bars recovery	Recovers stamina for respective player as they aren't attacking.	5:04	Pass
10	Round indicators displayed in top left and top right corner below health and stamina bars	Should fill with a colour based on who won the round. Winner of round gets box filled.	4:00	Pass
11	Timer count	Should decrease as fight goes on second by second	4:47	Pass
12	Timer reaches 0 (not final round)	Round should end and compare health levels of both warriors. Message to display this visually.	3:52	Pass
13	Health bar completely depleted	Round should end with player with remaining health being declared the winner.	4:22	Pass
14	Stamina bar reaches below critical level	Player with critical stamina should not be able to attack until it recovers stamina.	5:20	Pass
15	Streak counter displayed above player status bars	Updates once the match is over based on who won. Stored in separate file or database.	N/A	Fail
16	Player wins 2 rounds	Fight is over. Display streaks briefly and visually displays	6:31	No streak counters. Pass for all else.

		winner of match. Transition to game over state.		
--	--	--	--	--

Milestone 3: Paused

Test Number	What is being tested and relevant inputs	Expected Output	Timestamp	Pass/Fail
1	User input causing the playing game state to be paused	Pause any loops currently being run for the playing state on key press	7:13	Pass
2	Paused menu	When paused, a menu is displayed with options to resume or quit.	7:13	Pass
3	When user resumes. (from paused menu)	The paused loops should now be resumed, and the game state should now be back to the playing state.	7:18	Pass
4	When user quits. (from paused menu)	The game state should transition from paused menu to the main menu. Playing data is saved.	7:44	Pass

Milestone 4: Menu Interface

Test Number	What is being tested and relevant inputs	Expected Output	Timestamp	Pass/Fail
1	The background for the menu screen appears. Game transition from loading screen.	The background for the menu screen fills up the entire interface.	0:00	Pass
2	Buttons displayed correctly and in order.	Buttons appear on the menu.	7:59	Fail, buttons not implemented.
3	Mouse clicks on the GUI buttons.	Transition to the selected game state with relevant loading screens.	N/A	Fail. Not implemented.

Milestone 5: Warrior and map selection interface

Test Number	What is being tested and relevant inputs	Expected Output	Timestamp	Pass/Fail
1	Box prompt for entry of user surname before warrior selection.	Box appears in centre of screen prompting user to enter username.	N/A	Fail. Not implemented.
2	Keyboard presses to enter username in box prompt.	Key presses will display respective characters.	N/A	Fail. Not implemented.
3	Box prompt disappears upon user clicking enter key (if enough characters)	Box prompt disappears and player 2 is presented with box prompt.	N/A	Fail. Not implemented.
4	Box prompt does not disappear until valid username inputted including at least 4 characters and not already taken.	Box prompt stays until valid username input.	N/A	Fail. Not implemented.
5	Database registers valid entered username.	(back end) username added to database.	N/A	Fail. Not implemented.
6	Screen is split into two sides for player 1 and 2 on GUI.	Player 1 side on left, player 2 side on right, indicated by markers for player 1 and 2.	N/A	Fail. Not implemented.
7	Character interface displayed and mouse hovers on warrior images.	Animation to show selection, character summary and attributes appear.	N/A	Fail. Not implemented.
8	Mouse clicks on warrior images	Sound effect and cue as confirmation.	N/A	Fail. Not implemented.
9	Both warriors selected and confirmed.	Transition to map selection.	N/A	Fail. Not implemented.
10	Mouse clicks on map images.	Sound effect and animation. Transition to playing state.	N/A	Fail. Not implemented.

Milestone 6: Sound effects and background music

Test Number	What is being tested and relevant inputs	Expected Output	Timestamp	Pass/Fail
1	Background music for main menu state	Background music should play when player is navigating menu and selection.	10:16	Pass
2	Background music in the rounds	When entering the playing state intense background music should be played.	8:43	Pass
3	Background music in the final round	Music should intensify or change	9:35	Fail

4	Upon attack by either player	Sound effect upon attack depending on the warrior	8:46	Pass
5	On screen button press	Sound effect upon presses of these buttons	N/A	Fail. Not implemented.

Milestone 7: Game Over interface

Test Number	What is being tested and relevant inputs	Expected Output	Timestamp	Pass/Fail
1	Fight has ended	Game state should transition to game over state	6:40	Pass
2	Upon game over transition	Game over menu with 3 on screen buttons new match, rematch or save and quit.	6:40	Fail – No buttons. Prompts used instead.

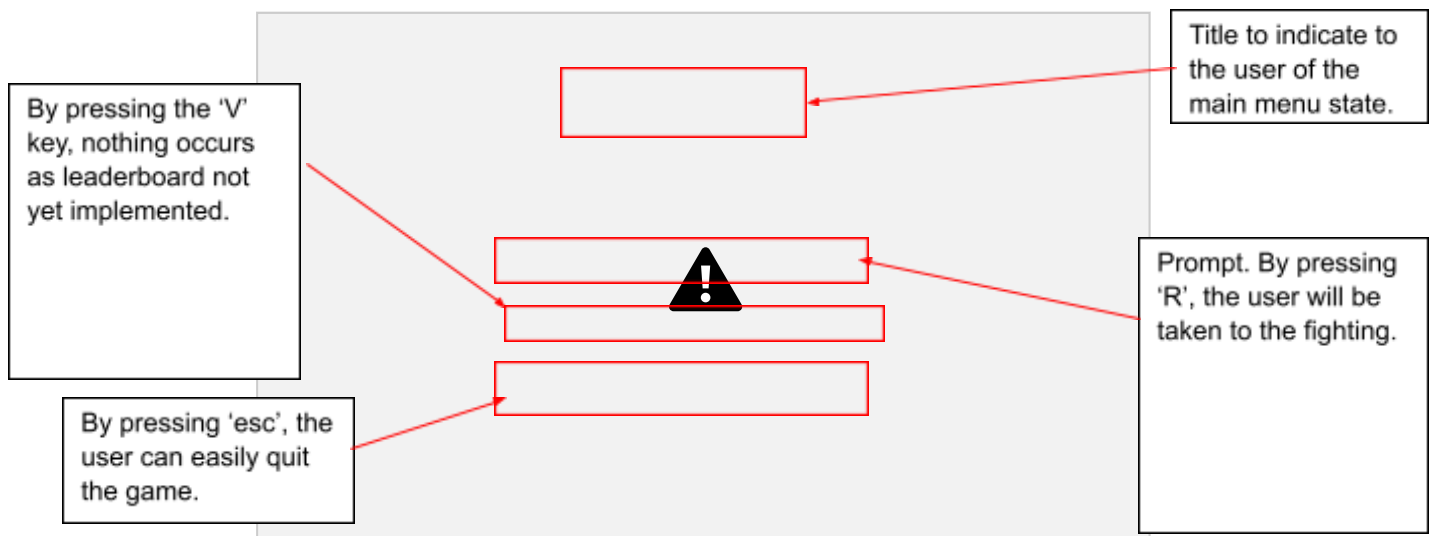
Milestone 8: Leaderboard display

Test Number	What is being tested and relevant inputs	Expected Output	Timestamp	Pass/Fail
1	Upon fight ended	Streak count should be stored in separate file or database	N/A	Fail. Not implemented.
2	Leaderboard interface	Should display visually, values stored in separate files or database.	N/A	Fail. Not implemented.
3	Leaderboard layout and ranking	Player with highest streak should be at the top. Username displayed next to streak.	N/A	Fail. Not implemented.

Usability features

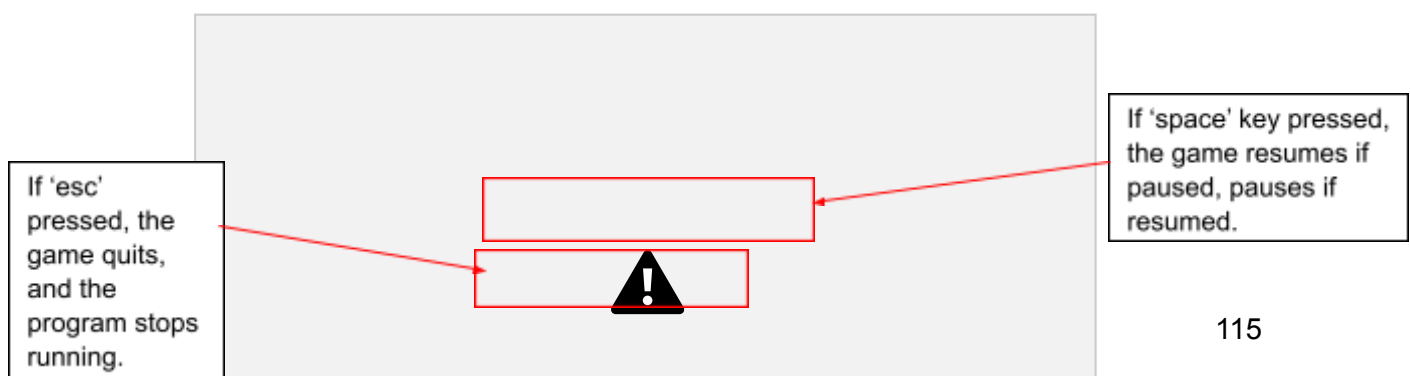
For the following, I have tested and explained these usability features and how they can be improved.

Navigation



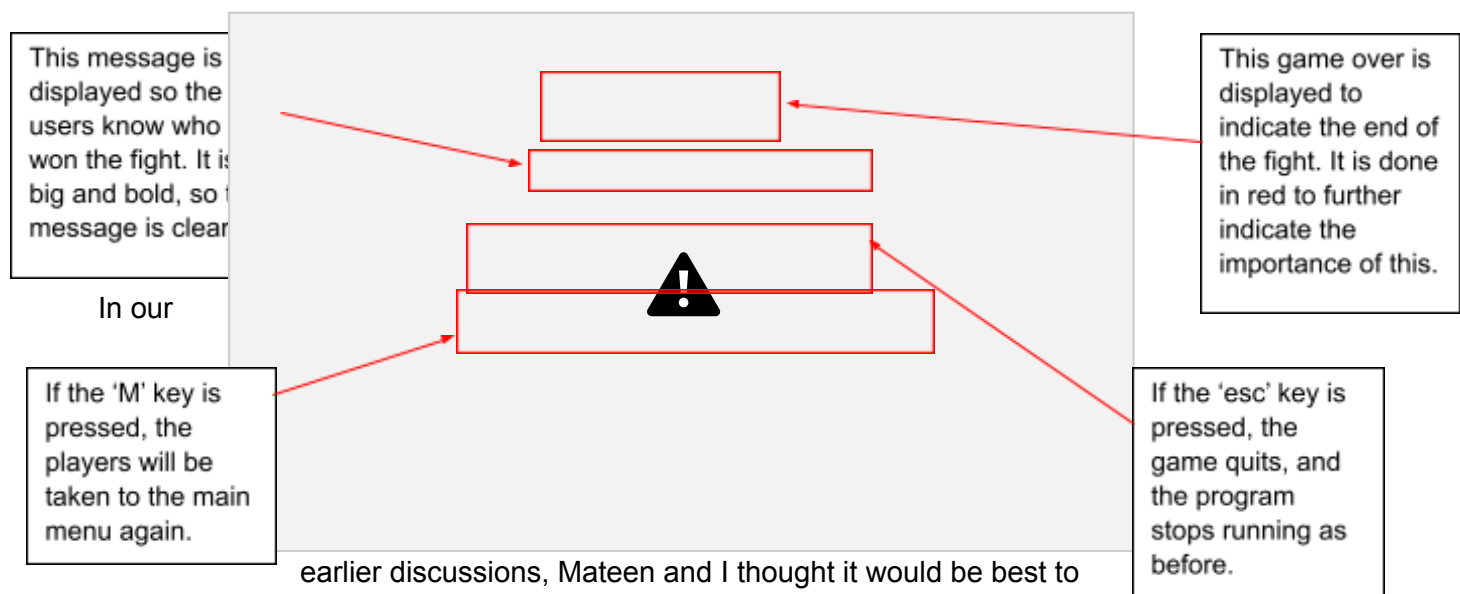
We discussed earlier, during the design phase, that there should be button on the main menu in order to transition between game states. Moreover, these should have taken mouse click inputs rather than prompts and keys. However, due to my time constraint and not being very familiar with Pygame graphical methods I was not able to implement this. This made me use keys rather than buttons to navigate the main menu. If I had more time, the problem could be fixed by using sprites for the buttons and implementing animations for each button. Moreover, the inputs registered would not be keys, but rather mouse clicks within the area of the button.

Mateen stated that he recalls the plan to use buttons in general for the navigation but has no problem with keys being used instead. He said that taking time constraints into account, using prompts and keys were clear enough but buttons were more intuitive and would have added a cleaner feel to the game.



When looking at the paused interface, initially the goal was to have buttons in order to navigate it, however, as with the main menu, due to my lack of knowledge in Pygame graphics and constraints, it was not possible for me to implement this. However, again, this can be fixed through more time and animations with sprites being implemented.

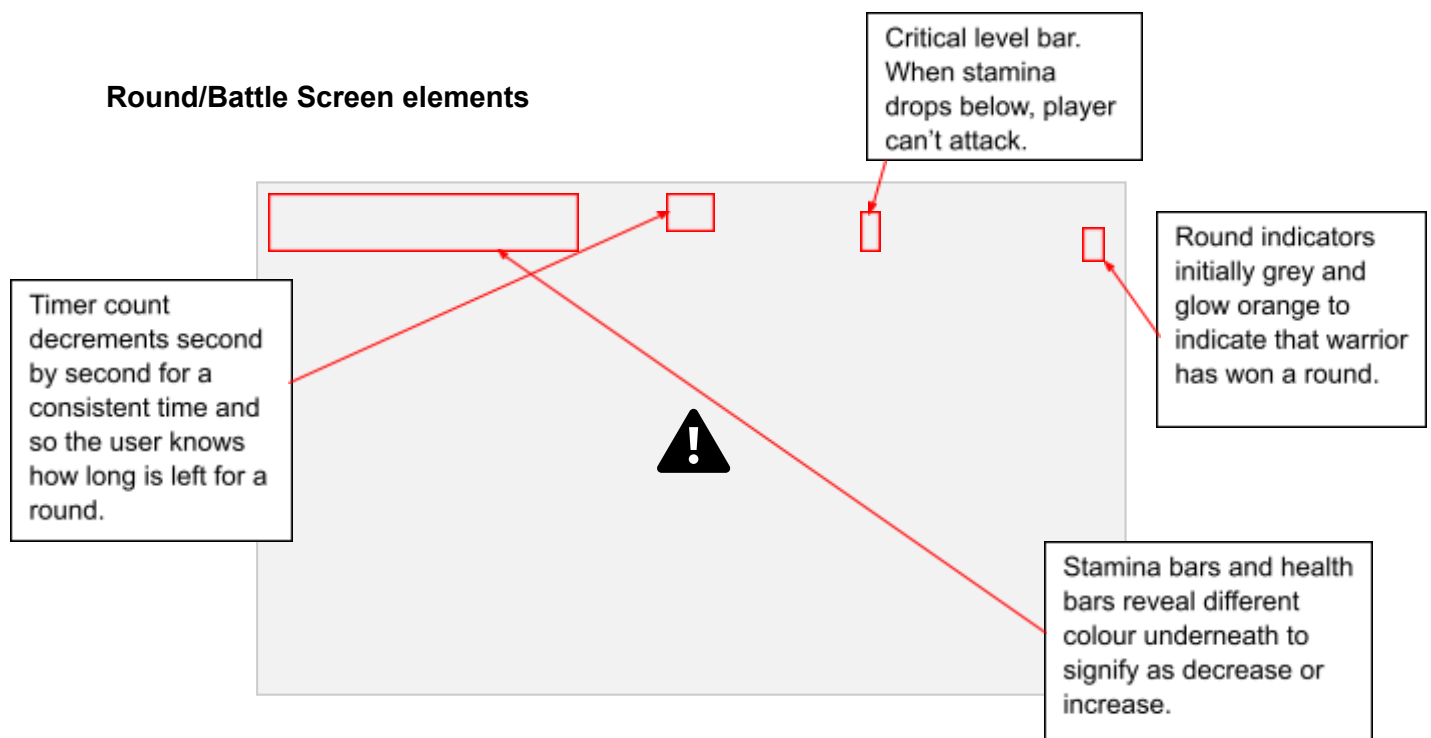
Mateen didn't have much to say but added that it was clear to navigate the menu with instructions. However, he did feel that the player would struggle to pause due to not knowing the key control for pause. This could be worked on by adding another section to the main menu which explained the game controls.



earlier discussions, Mateen and I thought it would be best to navigate with buttons and also have 3 options, to rematch, new match and quit and save. Most of this was due to the initial plan to have a leaderboard and warrior and map selection interface. This would add a layer of having different warriors and fighting styles, as well as fighting arenas. However, due to time constraints these options were not added and so were the buttons. The esc key is used throughout the game to quit which is consistent and gives an intuitive and predictable feel to the navigation which simplifies things. The game over message with the player won allows the users to know that the game is over and who won to indicate the fight has now ended.

The buttons as said before, can be worked on by using animations and sprites. However, the options to have a new match or quit and save, are dependent on having the selection interfaces which would need a decent amount of time to add these functionalities into the game. Also, the leaderboard would need to be added to save any data.

Mateen said that he felt that the interface was good enough and despite the lack of buttons, was easy to use and clear for the users. No further comments on this.



The round timer and round indicator indicate the status of the round and also the scoring in general. Whereas the health and stamina bars represent the status of the warriors within each round. This ensures that the player is aware of their warrior's health and stamina levels and also where they stand overall within the battle. Nonetheless, the stamina and health bars aren't visually pleasing due to not using sprites and the lack of two round indicators each side may confuse the user as to how many rounds the match consists of. This is because I didn't have the knowledge to implement fancier graphics for the bars and also due to me not being able to get one round indicator to light up at a time if two round indicators were used for each warrior. There are two fixes, either to include an explanation of the round format within the game or with more time, figure out the solution to getting two round indicators on each side working.

Mateen states that the critical level bar is particularly useful as it is clear and shows exactly when the player is too low on stamina and when they are able to attack again. However, the warriors not being animated make the game feel incomplete and not very visually pleasing or captivating as initially planned. He did add that he felt the game was somewhat playable however due to the clear and easy round system.

Controls

Overall, the controls for the warriors mid fighting are very simple as this allows for an easy learning curve. It follows a typical keyset for a two-player fighting game of W, A, D and UP, DOWN, LEFT which further simplifies and adds familiarity to the game. However, the game suffers from having no explanation of the controls unless briefed beforehand. Moreover, the

key press for pause is not explained either. This is an easy fix though, as an explanation of the controls can be added on the main menu.

Mateen said the controls were intuitive and very easy to use. But he said having an explanation would make the game far more usable to people who haven't played it before.

To what extent does the game meet the success criteria?

In this section, I will decide how much I have met each point in my success criteria and evaluate each point. In the next section, I will look at the points which may not have been fully met (limitations) and discuss improvement or new features to deal with these.

	Criterion	Justification	Met?	Evaluation
1	Menu screen displayed with clear buttons including Play, settings and quit.	This is the first game state the player will encounter. The user must be prepared to play before entering into a "war" between the fighters. This means being able to adjust settings before playing.	Partially met.	There is a menu screen but without the use of buttons. Instead, instructions and key presses are used due to my limited time. To implement buttons, I could use sprite sheets and an array in order to display a simple animation. Regarding the settings, I could use a different game engine or language as settings such as brightness, volume and sensitivity are advance and difficult implement with Pygame.
2	On menu screen: have a view leaderboard option	This allows for the players to view who has the longest streak of wins as well as to view their own high scores.	Partially met	While there is a view leaderboard option, it currently doesn't do anything as I decided to stop developing this after running into several complications. It would of course benefit from a working leaderboard system by storing scores stored in an external file and displaying those scores visually.
3	Display a Player and map selection screen	To allow for the users to select their characters before entering into a fight. Also, the users can choose a background of their fight.	Not met	There is no map or player selection as those features were omitted during development. This would require time and more knowledge in order to make a successful interface. Moreover, sprit sheets would need to be used and also more in detail profiles for the warriors. Child classes may need to be created to achieve this.

4	Display a box where each user enters a one-off username	So that the leaderboard can register a streak under an identifiable holder.	Not met	There is no username system as there is no leaderboard and hence no use since there is no leaderboard system. This features could be added by implementing a leaderboard system and using a while loop to only allow the fight to start when the users enter a valid username.
5	Give option for each player to select from at least 4 playable characters and 3 different maps	To provide variation in the game and allow for each player to experience a unique set of attributes and styles.	Not met	As there aren't sprite sheets or warrior profiles, this is not possible to implement until those are added into the game. I decided to focus more on the core mechanics as these extras could be worked on in future development.
6	Display each players health and stamina bar (stamina bar with critical level).	The player must know how much health and stamina he has left as these are the metrics which are essential to win the round or defeat the opponent.	Fully met	There are stamina and health bars clearly displayed at the top right and left corner of the screen. There is also a critical level bar. I could improve these by adding an animation which causes the stamina bar to flash when the critical level is reached.
7	Display round indicators	To show how many each player has won in the respective battle. This is crucial as to win a "war", the player must win 2 rounds.	Almost met	There are round indicators albeit one for each warrior. This is because I was not able to get the round indicators to light up properly when using two for each warrior, although this would have been more intuitive.
8	Display streak counter for each player	So that the player can know how many fights he has won consecutively and hence compete on leaderboard.	Not met	During development, I decided not to add these due to not being able to store these in a separate file. My user agreed that this would not be painful since these would not be stored or saved for future games. Also, no leaderboard.
9	Have a 30 second timer	To make the rounds restricted by time and the player can know where they stand in the round with regards to time	Fully met	There is a 30 second timer displayed at the top middle of the screen which decrements second by second.
10	Display each fighter in their starting position on either side of the screen.	To start the fight off with some distance and indicate the beginning of the round	Fully met	At the beginning of each round, the fighters are at the starting positions at a fixed distance apart.
11	Allow each fighter to move right, left and jump with separate keys on the keyboard	This movement should be gradual to challenge the players and add realism. Moreover, it is important as the player uses these	Fully met	The fighters can jump, move left and right by pressing keys on the keyboard. There are separate key sets for player 1 and 2. To improve this, animations and

		movements to dodge attacks, create space and combine with attack keys.		sprites could be used and hence combinations.
12	Allow each fighter to punch, kick, and block and for these to be combined with movement for more specific attacks	So that each player can control the way their fighter attacks the other player and can use combinations to score a high kick for example.	Not met	In the development phase, I ran out of time and was not able to add animations or specific attacks. This means that the attacks are a huge abstraction and cannot be combined with keys.
13	Register and show a depletion in stamina for every attack and recovery	To balance the game and make it more tactically orientated. This means players can't spam a barrage of attacks with no rest as this is unrealistic. If stamina bar drops below critical level, the player must recover beyond this before attacking.	Fully met	Stamina depletes on every attack and once it drops below the critical level, the player is no longer able to attack until this is recovered beyond this level. There is a steady recover which happens over time and happens more slowly within the critical region to further discourage users from spamming attacks. This is all displayed visually and also the critical level is indicated by a red bar.
14	Register and show a depletion in health for damage taken via attacks	As this allows for players to know how much damage they have dealt or received and hence change game plans.	Fully met	The health depletes correctly. This is also displayed visually. However, due to only one attack being available, the health depletion is constant and rather repetitive. This could be improved by introducing a multiplicity of attacks through combinations.
15	Round won if either health bar is depleted, or if time is up and one has higher health than the other	So that the indicators can be checked if this occurs, and the user can win the rounds and hence the battle. Player whose health bar is depleted loses the round. Player who has less health after time loses.	Fully met	If the health bar is depleted, the game registers that the player who last attacked and caused this, and hence decides the winner. If the timer runs out, the health level of each warrior is compared, and the warrior with the greater health wins the round.
16	"War" ends once either player has won 2 rounds	This is to signify that the fight is over. The warrior who won 2 rounds won the fight.	Fully met	The game transitions to the game over interface after either player has won two rounds. A message is displayed to show the winner.
17	Message asking for a rematch, new match, or exit	To allow for the player to build their streak and experience different fighting styles. Adds an element of competitiveness and variation.	Partially met	There are options to return to main menu or exit, however, not to rematch or start a new match. This is because initially, a leaderboard and map and warrior selection interface were going to be part of the solution. Hence, having these options made sense. However, due to none of

				these features being added, the options are limited and simpler.
18	Sound effects for blocking, punching, and kicking and taking damage	Sound effects will add depth to the game and make it more appealing to other senses. It can also give an indication as to what the opponent may be doing through sound and sight.	Partially met	Sound effects are added for attacking and jumping. However, due to not having many attacks or implementing blocking, there are no advanced sound effect for those. This was discussed previously, but I decided not to go ahead with adding such a wide range of attacks during development due to lack of animation.
19	Background music for menu screen and fight	Adding background music that varies based on map selection gives each map and fight a unique feel. In the tiebreaker round, more tense music will be played to increase anticipation and tension.	Almost met	Background music does play for the menu screen and during the fight, however it does not intensify for the final round. Moreover, there is only one track added due to me not having experience with the Pygame mixer and agreeing with my stakeholder that this was not an important use of development time.

Maintenance and Limitations

Maintenance

I have currently not implemented maintenance features within my game. However, if this game were at the stage of being distributed and released to the public, I would need to have regular updates, client feedback systems and a bug tracking system to ensure playability and high performance.

My game is currently not complete, and I have not added all the features I initially set out to add due to time constraints and limited knowledge. Upon completion I would like to add certain features such as a monitoring system within the game for any updates which I release so that the user is always playing on the latest and greatest version of the game. I would also like to incorporate a performance monitoring system to see how the game may be impacted by load. This is important due to the leaderboard which I will add constantly needing to update. Of course, some of these features depend on the success of the game.

Limitations of the solution and how they can be improved or changed

Below I have listed some limitations and explained how they could be improved or approached in future development.

Menu screen displayed with clear buttons including Play, settings and quit.	This is the first game state the player will encounter. The user must be prepared to play before entering into a "war" between the fighters. This means being able to adjust settings before playing.	Partially met.
---	---	----------------

In order to complete this criterion, I would use a sprite sheet for the buttons and extract frames, inserting into an array and then displaying them sequentially in order to cause an animation. When the mouse cursor is within the area of the button and it is clicked, the game should transition to the next state. This overall would ensure buttons are displayed and usable with mouse clicks.

On menu screen: have a view leaderboard option	This allows for the players to view who has the longest streak of wins as well as to view their own high scores.	Partially met
--	--	---------------

To achieve this, I will need to gain more knowledge and spend time implementing a system which stores streaks and score to an external database and correctly displays this on screen. When the view leaderboard option is clicked, then the most recently updated values should be displayed along with the leaderboard.

Display a Player and map selection screen	To allow for the users to select their characters before entering into a fight. Also, the users can choose a background of their fight.	Not met
Give option for each player to select from at least 4 playable characters and 3 different maps	To provide variation in the game and allow for each player to experience a unique set of attributes and styles.	Not met

To implement a player and map selection screen, I would trigger these states when the user clicks play. An interface using sprite sheets would appear and I can display at least 4 playable characters with a little drop-down menu with their names and portraits. I would possibly create a child class for each warrior so that I can have different profiles but slightly different move sets and attack damage. After selecting the players, I would then trigger the map selection screen, where at least 3 maps are shown. I would make it so when the mouse cursor hovers over the maps, it is shown more closely. This feedback would make it clearer and more intuitive to the user when selecting.

Display a box where each user enters a one-off username	So that the leaderboard can register a streak under an identifiable holder.	Not met
---	---	---------

In order to add this, the external database and leaderboard systems should be in place so that the username can be sent and stored in the database. This username would have a streak registered rather than the warrior itself having this streak. The box can be displayed,

and a while loop can be used to ensure the user cannot proceed until a valid username is entered (i.e. the box will be displayed continuously until something valid is entered).

Allow each fighter to punch, kick, and block and for these to be combined with movement for more specific attacks	So that each player can control the way their fighter attacks the other player and can use combinations to score a high kick for example.	Not met
---	---	---------

To complete this requirement, I would add a check for more buttons being pressed. So, I would add more attacks which occur on different button presses including E for punch, R for kick and F for block as an example. I would then also check for a combination using a conditional statement with an 'and' so if E and W are pressed, an uppercut punch is thrown. For this to make more sense, I would also make the regions where the attack lands more specific and realistic, and more or less damage would be done depending on the part of the body.

Message asking for a rematch, new match, or exit	To allow for the player to build their streak and experience different fighting styles. Adds an element of competitiveness and variation.	Partially met
--	---	---------------

To complete this criteria, I would add options simply to rematch or new match, rather than returning to the main menu. If the users click rematch, the warriors and map will stay the same, but the streaks would update, if new match is selected, then the users are taken to re select the map and warriors. But it would be more logical to have map and warrior selection, as well as streaks implemented so that these options make more sense.

Sound effects for blocking, punching, and kicking and taking damage	Sound effects will add depth to the game and make it more appealing to other senses. It can also give an indication as to what the opponent may be doing through sound and sight.	Partially met
Background music for menu screen and fight	Adding background music that varies based on map selection gives each map and fight a unique feel. In the tiebreaker round, more tense music will be played to increase anticipation and tension.	Almost met

The sound effects implementation being fully met depends on the extra attacks and combinations being added to the game. For each of these I would add a different attack using the simple Pygame mixer command. Moreover, when the player receives damage, I would add a sound to indicate impact and damage.

Regarding the background music, I would find out how to use conditionals along with the background music, and I would change or intensify the music when the warriors have both won one round (i.e. final round).

Additional limitation: Tie – health is equal for both players and timer reach 0

For this, I realised this limitation after the development phase and hence it was late to go back and make this implementation. However, I would use an if statement when the timer count reaches 0 to check if the health of warriors is equal, in which case neither warrior would win the round.

Overall summary and conclusion

To conclude, the projects core mechanics and round system functionality as well as the types of interfaces have been developed. It is currently not fully complete yet is robust enough to be playable. If worked on and further developed, such as the addition of animations, extra characters and even multiplayer modes or single player using AI, the game would be a lot more appealing and distributable. The base for a good game has been developed, though there is plenty of room for expansion.

Appendix

Bibliography

Ref: History of Street Fighter

Wiki: [https://en.wikipedia.org/wiki/Street_Fighter_\(video_game\)](https://en.wikipedia.org/wiki/Street_Fighter_(video_game))

Ref: Street Fighter game screenshots

<https://www.retrogames.cc/arcade-games/street-fighter-us-set-1.html>

Ref 2: Street fighter game screenshots

https://www.arcadequartermaster.com/capcom/sf1_bosses.html

Code listing

combatwarrior.py – main file

```
# importing libraries and resources
import pygame
import sys
from warriors import Warrior
#import sqlite3

# initialisation
pygame.init()

# Colors
RED = (255, 0, 0)
GREEN = (0, 255, 0)
LIGHT_BLUE = (4, 231, 231)
BLACK = (0, 0, 0)
GREY = (128, 128, 128)
LIGHT_GREY = (202, 204, 206)
ORANGE = (229, 83, 0)
WHITE = (255, 255, 255)

# Game window dimensions
SCRN_WIDTH, SCRN_HEIGHT = 1000, 600
scrn = pygame.display.set_mode((SCRN_WIDTH, SCRN_HEIGHT))
pygame.display.set_caption("Combat Warrior")

# Creating the clock object to cap my frame rate
clock = pygame.time.Clock()
FPS = 60

# original positions of warriors
w1_orgnl_pos_x = 200
w2_orgnl_pos_x = 700
orgnl_pos_y = 350

# loading background image
bg_img = pygame.image.load("Game
```

```
Assets/Backgrounds/nicebg.jpg")

# Creating instances of my warriors
warrior_1 = Warrior("Warrior 1", w1_orgnl_pos_x, orgnl_pos_y,
90, 190, (0, 0, 255))
warrior_2 = Warrior("Warrior 2", w2_orgnl_pos_x, orgnl_pos_y,
90, 190, RED)

# Adding sound and effects into my game
attack_sound = pygame.mixer.Sound("Game Assets/attack.wav")
jump_sound = pygame.mixer.Sound("Game Assets/jump.wav")
pygame.mixer.music.load("Game Assets/bg.mp3")
pygame.mixer.music.set_volume(0.5)
pygame.mixer.music.play(-1) # minus 1 means the music will
loop indefinitely.

# Important game variables
can_attack = False
start_countdown = 3
last_count = pygame.time.get_ticks()
round_countdown = 30

# Controlling game states using variables
paused = False
game_over = False
main_menu = True
username_entry = False
view_leaderboard = False

# Connecting to the leaderboard database
#conn = sqlite3.connect("Leaderboard/leaderboard.db")
#c = conn.cursor()

# Creating the table within the database
# c.execute('''CREATE TABLE leaderboard (username text, streak
int)''')

# Saving the changes in the database
#conn.commit()

# Closing connection
#conn.close()

def attack_controls():
    if event.type == pygame.KEYDOWN:
        # Check for key press events
        if event.key == pygame.K_e and warrior_1.stamina > 10:
            warrior_1.attack(scrn, warrior_2)
        elif event.key == pygame.K_p and warrior_2.stamina >
10:
```

```
        warrior_2.attack(scrn, warrior_1)
    elif event.key == pygame.K_w:
        warrior_1.make_jumping_possible()
    elif event.key == pygame.K_UP:
        warrior_2.make_jumping_possible()

# Function for drawing background
def draw_bg():
    bg = pygame.transform.scale(bg_img, (SCRN_WIDTH,
SCRN_HEIGHT)) # so that my image fits the screen properly
    scrn.blit(bg, (0, 0)) # blitting or putting my background
image on the game window

# Need to make my font
BIG_text_font = pygame.font.SysFont("Chiller", 70)
text_font = pygame.font.SysFont("Chiller", 50)
text_font_2 = pygame.font.SysFont("Chiller", 40)

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

# function draw text on to the paused screen
def paused_screen():
    draw_text("Paused", BIG_text_font, WHITE, 430, 70)
    draw_text("Press Space to Resume", text_font, WHITE, 340,
200)
    draw_text("Press esc to Quit", text_font, WHITE, 340, 280)

def game_over_screen():
    # transition to game over logic
    draw_text("Game Over", BIG_text_font, RED, 370, 70)
    draw_text("Press esc to Exit the game", text_font, WHITE,
290, 250)
    draw_text("Press M to return to main menu", text_font,
WHITE, 290, 320)
    if warrior_1.rounds_counter == 2:
        draw_text("Warrior 1 is Victorious!", text_font,
WHITE, 345, 150)
    else:
        draw_text("Warrior 2 is Victorious!", text_font,
WHITE, 330, 150)

def main_menu_screen():
```

```
draw_text("Main Menu", BIG_text_font, WHITE, 370, 70)
draw_text("Press R to start the game", text_font, WHITE,
290, 250)
draw_text("Press V to view leaderboard", text_font, WHITE,
290, 320)
draw_text("Press esc to exit the game", text_font, WHITE,
290, 390)

# function for main menu key presses
def menu_logic():
    # declaring global variables to be accessed.
    global main_menu
    global view_leaderboard
    # if main menu is true then rounds reset and so does the
    result.
    if main_menu:
        warrior_1.rounds_counter = 0
        warrior_2.rounds_counter = 0
        warrior_1.round_result = None
        warrior_2.round_result = None
        # makes main menu false and hence enters playing state
        if event.key == pygame.K_r:
            main_menu = False
        # quits the game.
        elif event.key == pygame.K_ESCAPE:
            pygame.quit()
            sys.exit()
        # elif event.key == pygame.K_v:
        # view_leaderboard = True

def game_over_logic():
    # declaring my global variables to be accessed.
    global game_over
    global main_menu
    # the logic only applies if game over is true.
    if game_over is True:
        # quits the game.
        if event.key == pygame.K_ESCAPE:
            pygame.quit()
            sys.exit()
        # returns to main menu.
        if event.key == pygame.K_m:
            game_over = False
            main_menu = True

# function for the pause logic. called in the game loop
def pause_logic():
    global paused # declaring my global variable, so it can
```



```
be accessed in my function.
    # if the key pressed on keydown is space, the pause logic
    is implemented.
    if event.key == pygame.K_SPACE:
        # if paused is currently true, set it back to false.
        if paused:
            paused = False
        # if not true, paused is now true.
        else:
            paused = True
    # this is allows me to quit the game if paused is true.
    elif event.key == pygame.K_ESCAPE and paused is True:
        pygame.quit()
        sys.exit()

# function for warrior movement controls
def warrior_movement():
    # Warrior 1 horizontal movement
    warrior_1.running = keys[pygame.K_d] or keys[pygame.K_a]
    if keys[pygame.K_d]:
        warrior_1.move_right()
    if keys[pygame.K_a]:
        warrior_1.move_left()

    # Warrior 2 horizontal movement
    # relating to moving right and left
    warrior_2.running = keys[pygame.K_RIGHT] or
keys[pygame.K_LEFT]
    if keys[pygame.K_RIGHT]:
        warrior_2.move_right()
    if keys[pygame.K_LEFT]:
        warrior_2.move_left()

    # Making warriors jump
    warrior_1.jump()
    warrior_2.jump()

# Function for drawing health bars
def draw_healthBars(health, x, y):
    # Calculate a ratio in order to multiply by the health bar
    width to show a depletion.
    ratio = health / 100
    # I have layered a few rectangle in order to show
    depletion clearly
    # rectangle for the border around the health bar
    pygame.draw.rect(scrn, BLACK, (x-2, y-2, 304, 19))
    # rectangle as a background to indicate depletion
    pygame.draw.rect(scrn, RED, (x, y, 300, 15))
    # Rectangle to show remaining health
```

```
pygame.draw.rect(scrn, GREEN, (x, y, 300 * ratio, 15))

# Function for drawing stamina bars
def draw_staminaBars(stamina, x, y):
    # Calculate a ratio in order to multiply by the stamina
    # bar width to show a depletion.
    stamina_ratio = stamina / 100
    # rectangle for the border around the stamina bar
    pygame.draw.rect(scrn, BLACK, (x - 2, y - 2, 304, 19))
    # rectangle as a background to indicate depletion
    pygame.draw.rect(scrn, GREY, (x, y, 300, 15))
    # Rectangle to show remaining stamina
    pygame.draw.rect(scrn, LIGHT_BLUE, (x, y, 300 *
stamina_ratio, 15))
    # Rectangle to indicate critical level
    pygame.draw.rect(scrn, RED, (x+300*0.1, y, 5, 15))

# function for round scoring and ending
def round_end(winner_name):
    global round_countdown
    # if warrior_1.round_won or warrior_2.round_won:
    if warrior_1.round_won:
        round_countdown = 30
        # This print statement was used to check if respective
        # warriors round counter was being updated
        draw_text(f"{winner_name} wins Round!", text_font,
(255, 255, 255), 340, 120)
        pygame.display.flip() # makes sure the display
updates with message immediately.
        pygame.time.delay(2000) # delay for 2 seconds before
resetting
        warrior_1.rounds_counter += 1
        warrior_1.reset_status(w1_orgnl_pos_x)
        warrior_2.reset_status(w2_orgnl_pos_x)
        print(str(warrior_1.rounds_counter))
    elif warrior_2.round_won:
        round_countdown = 30
        # This print statement was used to check if respective
        # warriors round counter was being updated
        draw_text(f"{winner_name} wins Round!", text_font,
(255, 255, 255), 340, 120)
        pygame.display.flip() # makes sure the display
updates with message immediately.
        pygame.time.delay(2000) # delay for 2 seconds before
resetting
        warrior_1.reset_status(w1_orgnl_pos_x)
        warrior_2.reset_status(w2_orgnl_pos_x)
        warrior_2.rounds_counter += 1
        print(str(warrior_2.rounds_counter))
```

```
def round_end_ontime():
    if warrior_1.health > warrior_2.health:
        warrior_1.round_won = True
        warrior_1.round_result = "won"
    else:
        warrior_2.round_won = True
        warrior_2.round_result = "won"

# Function for drawing round indicators
def draw_round_indicators(x, y, round_result):
    # Border around round indicator
    pygame.draw.rect(scrn, BLACK, (x - 2, y - 2, 19, 19))
    # updates the round indicators based on round result. This
    means it will stay on the screen after the reset.
    if round_result == "won":
        # Indicators will be drawn in dark gray. They will be
        filled with orange if won.
        pygame.draw.rect(scrn, ORANGE, (x, y, 15, 15))
        # print("Changed")
    else:
        # Indicators will be drawn in dark gray. They will be
        filled with orange if won.
        pygame.draw.rect(scrn, LIGHT_GREY, (x, y, 15, 15))

# this function will handle username entry and input
def enter_username():
    global username_entry
    user_1 = str(input(" User 1 Username: "))
    user_2 = str(input(" User 2 Username: "))
    warrior_1.username = user_1
    warrior_2.username = user_2
    username_entry = True

# This function will update the leaderboard
#def update_leaderboard(username, streak):
#    # global conn
#    # global c
#    # conn = sqlite3.connect("Leaderboard/leaderboard.db")
#    #c = conn.cursor()

#    # Checking if username exists in leaderboard
#    c.execute("SELECT * FROM leaderboard WHERE username=?",
#    (username,))
#    # existing_user = c.fetchone()

#    # If there is an existing user, we update their streak
```

```
# if existing_user:
#     new_streak = streak
#     c.execute("UPDATE leaderboard SET streak=? WHERE
username=?", (new_streak, username))
# If the player doesn't exist, insert them into
leaderboard
# else:
#     c.execute("INSERT INTO leaderboard VALUES (?, ?)",
(username, streak))

# # we must save these changes made in our database
#conn.commit()

# close the connection
#conn.close()

# This function will display the leaderboard
#def display_leaderboard():
#     global conn
#     global c

#     conn = sqlite3.connect("Leaderboard/leaderboard.db")
#     c = conn.cursor()

#     Fetch leaderboard data
#     c.execute("SELECT * FROM leaderboard ORDER BY streak
DESC")
#     leaderboard_data = c.fetchall()

#     Displaying the leaderboard
#     y_offset = 150
#     for i, (username, streak) in
enumerate(leaderboard_data[:5]): # to only display the top
five players
#         draw_text(f"{i+1}. {username}: {streak}", text_font,
WHITE, 200, y_offset)
#         y_offset += 50

#     Close connection
#     conn.close()

# Main game loop and event handler
running = True

# This keeps the game going until the user decides to exit
while running:
    # Setting my frame rate
    clock.tick(FPS)
```

```
# drawing bg
draw_bg()

# play background music

# event handler
for event in pygame.event.get():
    if event.type == pygame.QUIT:
        running = False

    elif event.type == pygame.KEYDOWN and can_attack is
True:
        # Check for key press events
        if event.key == pygame.K_e and warrior_1.stamina >
10:
            warrior_1.attack(scrn, warrior_2)
            attack_sound.play()
        elif event.key == pygame.K_p and warrior_2.stamina
> 10:
            warrior_2.attack(scrn, warrior_1)
            attack_sound.play()
        elif event.key == pygame.K_w:
            warrior_1.make_jumping_possible()
            jump_sound.play()
        elif event.key == pygame.K_UP:
            warrior_2.make_jumping_possible()
            jump_sound.play()

        # calling the paused function if mid round
        pause_logic()

        # Paused logic. Paused only set to true if space
is pressed.
        elif event.type == pygame.KEYDOWN:
            pause_logic()

        # Game Over logic
        game_over_logic()

        # main menu key presses
        menu_logic()

# paused screen
if paused is True:
    paused_screen()

# game over screen
elif game_over is True:
    game_over_screen()

# main menu screen
```

```
elif main_menu is True:
    main_menu_screen()

# username entry
# elif username_entry is False:
#     enter_username()

else:
    # drawing my warriors
    warrior_1.draw_warrior(scrn)
    warrior_2.draw_warrior(scrn)

    # Making warriors face each other
    warrior_1.face_correctly(warrior_2)
    warrior_2.face_correctly(warrior_1)

    # Warrior recovery
    warrior_1.recovery()
    warrior_2.recovery()

    # Drawing warrior status bars on the screen
    draw_health_bars(warrior_1.health, 30, 30)
    draw_health_bars(warrior_2.health, 670, 30)
    draw_stamina_bars(warrior_1.stamina, 30, 54)
    draw_stamina_bars(warrior_2.stamina, 670, 54)

    # Drawing round indicators
    draw_round_indicators(950, 74, warrior_2.round_result)
    draw_round_indicators(30, 74, warrior_1.round_result)

    # making the start counter go down
    if start_countdown > 0:
        can_attack = False
        if warrior_1.rounds_counter == 0 or
warrior_2.rounds_counter == 0:
            draw_text("On Guard...", text_font, WHITE,
420, 120)
            draw_text(str(start_countdown)+"!", text_font,
WHITE, 480, 170)
            count_timer = pygame.time.get_ticks()
            if count_timer - last_count > 1000:
                start_countdown -= 1
                last_count = count_timer
            elif warrior_1.rounds_counter == 1 and
warrior_2.rounds_counter == 1:
                draw_text("SUDDEN DEATH", text_font, RED, 380,
120)
                draw_text(str(start_countdown) + "!",
text_font, RED, 480, 170)
                count_timer = pygame.time.get_ticks()
                if count_timer - last_count > 1000:
```

```
        start_countdown -= 1
        last_count = count_timer

    # game locked until countdown is 0
    if start_countdown == 0:

        # drawing and updating my round timer.
        draw_text(str(round_countdown), text_font_2,
WHITE, 480, 20)
        count_timer = pygame.time.get_ticks()
        if count_timer - last_count > 1000:
            round_countdown -= 1
            last_count = count_timer

        can_attack = True

        # Making my warriors move on command
        keys = pygame.key.get_pressed()

        # warrior movement controls now accessible
        warrior_movement()

    # Ending the round
    if warrior_1.rounds_counter == 2 or
warrior_2.rounds_counter == 2:
        game_over = True
        # if warrior 1/2 win's round, reset the start
countdown and call the round end function
        if warrior_1.round_won:
            start_countdown = 3
            round_end(warrior_1.name)
        elif warrior_2.round_won:
            start_countdown = 3
            round_end(warrior_2.name)
        # if the round countdown reaches 0, implement the
logic of the round end on time function.
        elif round_countdown == 0:
            round_end_ontime()

        # updating the leaderboard
        #update_leaderboard(warrior_1.username,
warrior_1.streak)
        #update_leaderboard(warrior_2.username,
warrior_2.streak)

    # updating the display
    pygame.display.update()

# exiting pygame
pygame.quit()
sys.exit()
```

warriors.py – class file

```
# Importing libraries and resources
import pygame

# Initialisation
pygame.init()

class Warrior:

    # Constructor method
    def __init__(self, name, x, y, width, height, color):
        self.name = name
        self.rect = pygame.Rect(x, y, width, height)
        self.color = color
        self.flip = True
        self.velocity = 5
        self.is_jumping = False
        self.jump_height = 10

        # relating to rounds and scoring logic
        self.health = 100
        self.stamina = 100
        self.is_attacking = False
        self.running = False
        self.round_won = False
        self.round_result = None
        self.rounds_counter = 0

        # relating to leaderboard
        self.username = None
        self.streak = 0 # this attribute will be updated upon
a match won

    # This function makes my warrior go left
    def move_left(self):
        self.rect.x -= self.velocity
        # This stops my warrior going of the left side of the
screen
        if self.rect.left < 0:
            self.rect.x += self.velocity

    # This function makes my warrior go right
    def move_right(self):
        self.rect.x += self.velocity
        # This stops my warrior going of the right side of the
screen
```



```

        if self.rect.right > 1000:
            self.rect.x -= self.velocity

    # This function sets my attribute is jumping to true so
the jump method works.
    def make_jumping_possible(self):
        self.is_jumping = True

    # This function makes my warrior jump
    def jump(self):
        if self.is_jumping is True:
            if self.jump_height >= -10:
                grad = 1
                if self.jump_height < 0:
                    grad = -1
                self.rect.y -= (self.jump_height * 0.75)**2 *
grad
                self.jump_height -= 1
            else:
                self.is_jumping = False
                self.jump_height = 10

    # Makes sure the warriors always face each other.
    def face_correctly(self, target):
        # If the warrior being targeted centre is at an x
coordinate less than the attackers centre x coordinate we need
        # to flip
        if target.rect.centerx < self.rect.centerx:
            self.flip = True
        else:
            self.flip = False

    def attack(self, surface, target):
        self.is_attacking = True
        # Creating a rectangle which will be drawn upon key
press
        attacking_rect = pygame.Rect(self.rect.centerx - (2 *
self.rect.width * self.flip), self.rect.y,
                                     self.rect.width * 2,
self.rect.height)
        pygame.draw.rect(surface, (0, 255, 0), attacking_rect)
        # I have made it so that the stamina will only reduce
if the value after reduction will be positive. If
        # not the value will go to 0.
        if self.stamina - 10 > 0:
            self.stamina -= 10
        elif self.stamina - 10 < 0:
            self.stamina = 0
        # This means that the attack should only be registered
provided the attacking rectangle collides with
        # warrior

```

```
        if attacking_rect.colliderect(target.rect):
            # If the attack lands successfully, the target
should lose a set health.
            if target.health > 7.5:
                target.health -= 7.5
            elif target.health <= 7.5:
                target.health = 0
            # if targets health is 0, attacker won the
round.

            self.round_result = "won"
            self.round_won = True

        self.is_attacking = False

# Function to reset health and stamina
def reset_status(self, original_pos_ofx):
    self.health = 100
    self.stamina = 100
    self.rect.x = original_pos_ofx
    self.round_won = False

# Function for recovering stamina
def recovery(self):
    # if we are not attacking and stamina is less than 100
we recover stamina.
    if not self.is_attacking and 10 < self.stamina <= 100:
        self.stamina += 0.05
        # in order to cap the stamina level to 100.
        self.stamina = min(self.stamina, 100)
    elif self.is_attacking is False and self.stamina <=
10:
        self.stamina += 0.02

# Drawing my warrior to the screen
def draw_warrior(self, surface):
    pygame.draw.rect(surface, self.color, self.rect)
```